

**UNIVERSIDAD AUTONOMA DE MADRID**

**ESCUELA POLITECNICA SUPERIOR**



**Grado en Ingeniería Informática**

## **TRABAJO FIN DE GRADO**

**CHATBOT PARA ACCESO Y RECUPERACIÓN DE DATOS  
ABIERTOS GUBERNAMENTALES**

**Jesús Viejo Tardío**  
**Tutor: Iván Cantador Gutiérrez**

**FEBRERO 2021**



# **CHATBOT PARA ACCESO Y RECUPERACIÓN DE DATOS ABIERTOS GUBERNAMENTALES**

**AUTOR: Jesús Viejo Tardío**  
**TUTOR: Iván Cantador Gutiérrez**

**Departamento de Ingeniería Informática**  
**Escuela Politécnica Superior**  
**Universidad Autónoma de Madrid**  
**Febrero de 2021**



# Resumen

Este Trabajo Fin de Grado tiene como objetivo presentar un chatbot que facilite el acceso a los datos abiertos gubernamentales a los ciudadanos y otros actores.

Los datos abiertos son comúnmente publicados como archivos con datos estructurados en formatos estandarizados como CSV, XML o RDF, que requieren un gran esfuerzo para poder consultarlos por un usuario no experto. Este chatbot no solo permite buscar colecciones de datos, sino también explorar información dentro de estas. Esta exploración es realizada a través de unas consultas SQL que son fácilmente creadas por usuarios no expertos a través de una conversación en lenguaje natural. En particular, para la realización de este proyecto se han empleado los datos abiertos del Ayuntamiento de Madrid.

El sistema está compuesto por tres elementos: un agente de DialogFlow integrado en la plataforma de mensajería Telegram, que gestiona la interacción entre el chatbot y el usuario; el servidor web Spring Boot desplegado en Heroku, que implementa la lógica de negocio y de acceso a datos; y la base de datos relacional PostgreSQL que almacena las colecciones a consultar y los datos necesarios del sistema.

Este documento recoge el diseño y el desarrollo de estos tres elementos, satisfaciendo la flexibilidad, escalabilidad y mantenimiento del sistema para futuras integraciones, actualizaciones y modificaciones.

Se realiza y detalla un estudio dirigido a evaluar el chatbot según una serie de valores de los servicios públicos, además de medir distintas métricas objetivas y subjetivas. Los resultados experimentales muestran que el sistema propuesto supera a los métodos tradicionales de acceso a datos abiertos en los portales.

## Palabras clave

Chatbot, Datos Abiertos, E-Gobierno, Gobierno Abierto



# Abstract

This Bachelor Thesis aims to present a chatbot that facilitates access to open government data to citizens and other stakeholders.

Open data is commonly published as files with structured data in standardized formats such as CSV, XML, or RDF, which require a lot of effort to be consulted by a non-expert user. This chatbot not only allows you to search for collections of data, but also to explore information within these. This exploration is done through SQL queries that are easily created by non-expert users through a natural language conversation. In particular, the open data of the City of Madrid have been used to carry out this Project.

The system consists of three elements: a DialogFlow agent integrated into the Telegram messaging platform, which manages the interaction between the chatbot and the user; Spring Boot web server deployed in Heroku, which implements business and data access logic; and the PostgreSQL relational database that stores the collections to be queried and the necessary system data.

This document includes the design and development of these three elements, satisfying the flexibility, scalability and maintenance of the system for future integrations, upgrades and modifications.

A study is conducted and detailed aimed at evaluating the chatbot based on a number of public services values, in addition to measuring different objective and subjective metrics. Experimental results show that the proposed system outperforms traditional methods of accessing open data on portals.

# Keywords

Chatbots, Open Data, E-Government, Open Government





## **Agradecimientos**

A mi tutor Iván, por su apoyo y darme la oportunidad de realizar este proyecto.

A todas las personas, en especial a mi familia y pareja, y a aquellos que ya no están para celebrarlo, por todo su apoyo y dedicación durante esta aventura.



# ÍNDICE DE CONTENIDOS

<b>1 INTRODUCCIÓN</b>	<b>1</b>
1.1 MOTIVACIÓN	1
1.2 OBJETIVOS	1
1.3 ORGANIZACIÓN DE LA MEMORIA	2
<b>2 ESTADO DEL ARTE</b>	<b>3</b>
2.1 CHATBOTS	3
2.2 ESTUDIO DE LAS TECNOLOGÍAS UTILIZADAS	5
2.2.1 <i>Plataforma de mensajería</i>	5
2.2.2 <i>Herramientas de desarrollo de chatbots con motores IA</i>	6
2.2.3 <i>Servicio web</i>	7
2.2.4 <i>Base de datos relacional</i>	7
<b>3 DISEÑO</b>	<b>9</b>
3.1 ESQUEMA GENERAL INTERACCIÓN	9
3.2 DISEÑO DE LA INTERACCIÓN	10
3.2.1 <i>DialogFlow</i>	11
3.2.2 <i>Servidor web</i>	14
3.2.3 <i>Base de datos</i>	15
<b>4 DESARROLLO</b>	<b>21</b>
4.1 ESTRUCTURA DEL FLUJO	21
4.1.1 <i>Intent Welcome</i>	22
4.1.2 <i>Intent Help</i>	22
4.1.3 <i>Intent Get Collections</i>	22
4.1.4 <i>Intent Filter Keywords Collections</i>	22
4.1.5 <i>Intent Select Collection</i>	23
4.1.6 <i>Intent Get Collection Fields</i>	23
4.1.7 <i>Intent Select Without Where</i>	25
4.1.8 <i>Intent Select With Where</i>	26
4.1.9 <i>Intent Satisfactory Response</i>	26
4.2 ARQUITECTURA DEL SERVIDOR WEB	26
4.2.1 <i>Capa de presentación y controlador</i>	27
4.2.2 <i>Capa de negocio</i>	29
4.2.3 <i>Capa de acceso a datos</i>	30
<b>5 INTEGRACIÓN, PRUEBAS Y RESULTADOS</b>	<b>33</b>
5.1 EXPERIMENTO	33
5.1.1 <i>Metodología de evaluación</i>	33
5.1.1 <i>Métricas de evaluación</i>	35
5.2 RESULTADOS	35
5.2.1 <i>Eficacia y eficiencia</i>	35
5.2.2 <i>Usabilidad</i>	37
5.2.3 <i>Valores de los servicios públicos</i>	37
<b>6 CONCLUSIONES Y TRABAJO FUTURO</b>	<b>39</b>
6.1 CONCLUSIONES	39
6.2 TRABAJO FUTURO	39
<b>REFERENCIAS</b>	<b>41</b>
<b>GLOSARIO</b>	<b>43</b>
<b>ANEXOS</b>	<b>I</b>
A DETALLE WEBHOOK REQUEST Y WEBHOOK RESPONSE	I
B EJEMPLO DE CONVERSACIÓN CON EL BOT	III
C EJEMPLO INTERACCIÓN CON EL PORTAL	VIII
D INTEGRACIÓN EN TELEGRAM	X



## ÍNDICE DE FIGURAS

FIGURA 3-1: ESQUEMA GENERAL DE LOS COMPONENTES .....	9
FIGURA 3-2: DIAGRAMA ER DE LA BASE DE DATOS .....	20
FIGURA 4-1: DIAGRAMA DE FLUJO DE CONVERSACIÓN .....	21
FIGURA 5-1: NIVELES DE DIFICULTAD DE LAS TAREAS PERCIBIDOS POR LOS PARTICIPANTES.....	37
FIGURA 5-2: OPINIÓN SOBRE LOS VALORES DE LOS SERVICIOS PÚBLICOS DEL PORTAL (P) Y EL CHATBOT (C) .	38

## ÍNDICE DE TABLAS

TABLA 3-1: DESCRIPCIÓN INTENTS .....	12
TABLA 3-2: FRASES DE ENTRENAMIENTO DE LOS INTENTS .....	13
TABLA 3-3: DETALLE TABLA USUARIO .....	16
TABLA 3-4: DETALLE TABLA COLECCIÓN .....	16
TABLA 3-5: DETALLE TABLA KEYWORD .....	16
TABLA 3-6: DETALLE TABLA LOG.....	17
TABLA 3-7: DETALLE TABLA CALENDARIO _MADRID.....	17
TABLA 3-8: DETALLE TABLA RETRIBUCIÓN _MADRID .....	18
TABLA 3-9: DETALLE TABLA DECIDE _MADRID .....	19
TABLA 3-10: DETALLE TABLA PRESUPUESTO _GENERAL _MADRID .....	19
TABLA 4-1: TIPOS DE CONSULTAS SOPORTADAS .....	24
TABLA 4-2: RELACIÓN INTENTS - PARÁMETROS .....	28
TABLA 5-1: ÍTEMS DEL CUESTIONARIO DIRIGIDO A EVALUAR LOS VALORES DE LOS SERVICIOS PÚBLICOS [9]..	34
TABLA 5-2: VALORES DE EFICACIA Y EFICIENCIA PARA LAS TAREAS EN EL PORTAL Y HOJA DE CÁLCULO.....	36
TABLA 5-3: VALORES DE EFICACIA Y EFICIENCIA PARA LAS TAREAS EN EL CHATBOT .....	36



# 1 Introducción

---

Esta sección muestra una visión del marco general del proyecto, indicando la motivación para la realización de este, los objetivos que se quieren lograr con su desarrollo y la estructura del documento.

## 1.1 Motivación

Esta memoria de TFG tiene como propósito describir el objetivo, diseño, desarrollo y funcionamiento del chatbot creado para este proyecto, un chatbot que facilite el acceso a los datos abiertos gubernamentales a los ciudadanos y otros actores.

El Gobierno Abierto<sup>1</sup> es una forma de gobernanza de las Administraciones que busca fortalecer la democracia a través de un gobierno colaborativo y participativo [1]. Un gobierno transparente fomenta y promueve la rendición de cuentas al público y proporciona información sobre lo que está haciendo y sus planes de acción. Permitir a los ciudadanos llevar a cabo un control de las actividades del gobierno, creando valor económico y social. Permite la cooperación con la ciudadanía, empresas, asociaciones y otros actores, ayudando a la administración pública a beneficiarse del conocimiento y experiencia de estos. Por tanto, aumenta el protagonismo y la implicación de los ciudadanos en los asuntos públicos, y compromete a los actores políticos con los ciudadanos [2].

En los últimos años, siguiendo las políticas de Gobierno Abierto, y para dotar de mayor transparencia a las Administraciones Públicas, se han publicado los datos que poseen estas en formato abierto. Los datos abiertos<sup>2</sup> son aquellos que cualquiera es libre de utilizar, reutilizar y redistribuir, sin restricciones de acceso, patentes, copyright u otros mecanismos de control. La reutilización de estos datos consiste en actividades de copia, difusión, modificación, extracción o reordenación de la información, por parte tanto de personas físicas como jurídicas. Tiene como objetivo facilitar el desarrollo de nuevos productos, servicios y soluciones de alto valor socioeconómico, además de permitir un alto grado de transparencia y rendición de cuentas.

Sin embargo, esta tarea de reutilización puede llegar a ser tediosa y compleja para un usuario no experto, que debe buscar las colecciones, leer sus descripciones, descargarlas y analizar su contenido.

## 1.2 Objetivos

Este proyecto intenta proponer una solución a este problema, facilitando la búsqueda y recuperación de la información de colecciones de datos abiertos, a través de un sistema conversacional con acceso mediante lenguaje natural.

Dentro de este objetivo principal, se dividen objetivos más concretos:

---

<sup>1</sup> Gobierno Abierto , [https://administracionelectronica.gob.es/pae\\_Home/pae\\_Estrategias/pae\\_Gobierno\\_Abierto\\_Inicio.html](https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/pae_Gobierno_Abierto_Inicio.html)

<sup>2</sup> Datos abiertos, [https://administracion.gob.es/pag\\_Home/espanaAdmon/Transparencia-y-datos-abiertos-2/datos\\_abiertos.html](https://administracion.gob.es/pag_Home/espanaAdmon/Transparencia-y-datos-abiertos-2/datos_abiertos.html)

- Proporcionar una interacción entre el usuario y el chatbot a través de un lenguaje formal, además del uso sencillo de este. Utilizando para esta interacción una plataforma de mensajería instantánea.
- Diseñar la aplicación permitiendo un fácil mantenimiento y escalabilidad, para poder añadir nuevas colecciones a consultar, con un mínimo esfuerzo y sin cambio de arquitectura y desarrollo.
- Optimizar la comunicación entre los componentes para una correcta usabilidad.
- Implementar el caso de estudio en las colecciones del portal de datos abiertos del Ayuntamiento de Madrid <sup>3</sup>.
- Evaluar el proyecto mediante un estudio de usuarios, comparando el uso del chatbot (ver Anexo B) con el uso del portal y hojas de cálculo (ver Anexo C).

### 1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- *Estado del arte*: Se describe la definición del término chatbot, explicando los diferentes sistemas conversacionales que existen y destacando las diferencias entre el chatbot y un asistente virtual.  
Se realiza también un análisis de los diferentes componentes y las tecnologías que integra la aplicación.
- *Diseño*: Se explica en detalle cada uno de los componentes que forman parte de la aplicación, analizando los requisitos y el diseño final implementado. Se describen todos los intents que van a formar parte de DialogFlow; la arquitectura de Tres Capas del servidor web; y todas las entidades que forman la base de datos relacional.
- *Desarrollo*: Se detalla la estructura de flujo que sigue el chatbot, explicando el caso de uso de cada intent. Se describe la funcionalidad desarrollada del servidor, la paquetización seguida y el pseudocódigo de los métodos más importantes del sistema.
- *Pruebas y resultados*: Se explica el experimento llevado a cabo y los resultados obtenidos. Detallando la metodología y métricas estudiadas.
- *Conclusiones y trabajo futuro*: Se extraen las conclusiones de los resultados del estudio y se lleva a cabo un análisis de estas, observando puntos mejorables del chatbot y exponiéndolos para futuros trabajos.
- *Anexos*: Se añaden cuatro anexos: detallando la información del Webhook Request y Webhook Response (Anexo A); adjuntando una conversación entre un usuario y el chatbot (Anexo B); adjuntando una interacción entre un usuario y el portal de datos abiertos <sup>3</sup>, mostrando un ejemplo de estos datos (Anexo C); y explicando la integración del chatbot en Telegram (Anexo D).

---

<sup>3</sup> Portal datos abiertos Ayuntamiento Madrid, <https://datos.madrid.es/portal/site/egob/>



## 2 Estado del arte

---

En esta sección se realiza una contextualización sobre el sistema conversacional que vamos a desarrollar en este proyecto, el chatbot. Se analizan las diferencias con el otro sistema conversacional con capacidad para interpretar el lenguaje de un ser humano, el asistente virtual. Se detalla el estudio de las diferentes tecnologías elegidas para llevar a cabo el proyecto.

La Era Digital o la Era de la Información, es el nombre que recibe el período de la historia que transcurre desde la revolución digital, muy ligado a las Tecnologías de la Información y la Comunicación (TIC). Actualmente, los expertos siguen debatiendo sobre la corrección etimológica de este término, siendo debate Era de la Información, o Era del Conocimiento. En España en el año 2019, el 90.7% de la población entre 16 y 74 años ha utilizado Internet en los últimos tres meses <sup>4</sup>, es decir, han tenido acceso a información o conocimiento. Y es en este detalle, donde radica el principal problema de dicha Era; existen casi infinitas referencias de información al alcance de un clic, pero resulta una tarea verdaderamente exigente y costosa para una persona poder consultar toda esta información. Este concepto se denomina Explosión de la Información.

Esta Explosión provoca un exceso de datos que debemos verificar, clasificar y relativizar. El estudio del Big Data o Macrodatos atenúa este problema. Además, surgen varias herramientas para facilitar el acceso a estos datos.

Los chatbots son un ejemplo de nuevas tecnologías que han aumentado considerablemente en los últimos años por cuatro grandes factores:

- La velocidad del tiempo de respuesta, del orden de milisegundos.
- Ofrecen servicio a multitud de usuarios de manera simultánea.
- Son capaces de identificar, reconocer, comprender, evaluar y responder cuestiones elaboradas de forma natural.
- Su especificación en la tarea a realizar. Un chatbot realiza una tarea muy específica.

### 2.1 Chatbots

Etimológicamente, chatbot, es un sustantivo formado por la unión de dos lexemas, Chat y (Ro)Bot. Se puede definir como robot de conversación diseñado para simular una conversación inteligente con uno o varios usuarios a través de métodos textuales o auditivos. Estos mantienen la conversación a través de las aplicaciones de mensajería, como Facebook Messenger, Telegram, Twitter o Slack.

Los chatbots pueden integrar la inteligencia artificial (IA) para aprender de las interacciones con los usuarios libremente. Como no siempre es este el objetivo, otros utilizan la inteligencia artificial de manera más controlada. Un caso muy reconocido fue el

---

<sup>4</sup> Población que usa Internet (en los últimos tres meses). Tipo de actividades realizadas por Internet, [https://www.ine.es/ss/Satellite?L=es\\_ES&c=INESeccion\\_C&cid=1259925528782&p=1254735110672&pagename=ProductosYServicios%2FPYSLayout](https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259925528782&p=1254735110672&pagename=ProductosYServicios%2FPYSLayout)

chatbot desarrollado por Microsoft en 2016 Tay <sup>5</sup> que, tras mantener unas horas de interacción con los usuarios, desarrolló comportamientos racistas y xenófobos.

Según su grado de interacción con los usuarios, pueden clasificarse en:

- *Chatbots dirigidos*: Aquellos que ofrecen unas preguntas predeterminadas a los usuarios a través de algunos elementos fijos, por ejemplo, botones. De esta manera, la interacción con el usuario será muy controlada. Por ejemplo, el chatbot desarrollado por el popular portal Chollometro <sup>6</sup>.
- *Chatbots conversacionales*: Aquellos que ofrecen una libertad a las preguntas que el usuario quiere formular. Este tipo de chatbots mantiene una interacción con el usuario mucho más natural. Por ejemplo, el chatbot del banco BBVA para sus clientes, Blue <sup>7</sup>.

En cuanto a la finalidad del chatbot, existen multitud de aplicaciones prácticas:

- *Servicio de mejora operativa de procesos*: Se procura reducir el tiempo destinado a una tarea. Este proyecto se encasilla en esta finalidad.
- *Servicio de atención al cliente*: Se procura atender en una jornada ininterrumpida y sin esperas. Actualmente, algunos centros médicos realizan la citación para una consulta médica con estos servicios. También se procura dar una atención a preguntas frecuentes por parte de los clientes.
- *Servicio de comunicación y marketing*: Se procura dar un servicio de consultas dentro de una aplicación móvil o web. Por ejemplo, las entidades bancarias los integran para ayudar a los usuarios a navegar por sus webs o aplicaciones móviles.
- *Servicio de ocio*: Se procura dar un servicio de entretenimiento, y establecer conversaciones naturales sin ninguna temática de negocio. Victoria <sup>8</sup>, desarrollado por el Ayuntamiento de Málaga con el objetivo de informar sobre la ciudad y darla a conocer.

Con esta definición, intuimos asistentes tan conocidos como Alexa (Amazon), Siri (Apple), Cortana (Windows) o Google Assistant (Google), aunque se debe diferenciar entre estos términos: chatbot y asistente virtual.

En los últimos años estos sistemas han tenido un aumento de popularidad y apoyo por parte de las organizaciones muy importante. Aunque en un principio y en grandes rasgos pueden entenderse de manera muy similar, son sistemas con ciertas diferencias:

- Mientras los chatbots están pensados para satisfacer a preguntas y respuestas en un ámbito más reducido y preparado, los asistentes tienen una plataforma interactiva más sofisticada, y un ámbito de interacción mucho más general y global.
- La interacción con los chatbots suele ser más corta, y con un objetivo muy concreto.
- Los asistentes virtuales se centran en el procesamiento del lenguaje general (PNL) y la comprensión del lenguaje natural (NLU). Gracias a esto, su grado de interacción es más conversacional, pudiendo comprender de mejor manera conversaciones cotidianas.

---

<sup>5</sup> Tay, <https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/>

<sup>6</sup> Chatbot Chollometro, <https://t.me/chollometro>

<sup>7</sup> Blue, <https://www.bbva.es/personas/banca-online/chatbots.html>

<sup>8</sup> Victoria, <https://googlechatbots.es/victoria/>

- Los chatbots, y más concretamente las herramientas de desarrollo con motores IA, por ejemplo, DialogFlow, se centran en el análisis de la frase y la extracción de palabras claves que ayuden a identificar la intención del usuario.

## 2.2 Estudio de las tecnologías utilizadas

La aplicación desarrollada está compuesta de un framework formado por los siguientes cuatro componentes:

- Plataforma de mensajería (Telegram).
- Herramienta de desarrollo de chatbots con motores IA para el procesamiento del lenguaje natural (DialogFlow).
- Servicio web o servidor (Heroku y Spring Boot).
- Base de datos relacional PostgreSQL (Heroku Postgres).

Para la elección de estos componentes, se consideraron varios aspectos y restricciones técnicas que se detallan a continuación.

### 2.2.1 Plataforma de mensajería.

Las plataformas de mensajería permiten establecer una comunicación instantánea y en tiempo real a través de internet, para el envío de texto, imágenes o cualquier otro tipo de documento entre dos o más usuarios.

Estas plataformas fueron utilizadas el pasado 2019 por más del 90% de población de 16 y 74 años <sup>4</sup>. Esto, unido a la facilidad de uso, hace que este servicio haya sido seleccionado para crear multitud de chatbots.

WhatsApp es el servicio de mensajería más utilizado en España <sup>9</sup>, aunque también tienen gran popularidad Telegram o Facebook Messenger:

- *Facebook Messenger* <sup>10</sup>: Plataforma de Facebook que permite interactuar de forma privada con los usuarios de la aplicación. Permite enviar mensajes de texto, imágenes y documentos.
- *WhatsApp* <sup>11</sup>: Aplicación que permite enviar mensajes de texto, imágenes y documentos, se pueden enviar stickers, ubicaciones, o realizar llamadas de voz y video. Puede ser usada tanto desde un dispositivo móvil, como desde un ordenador personal. Además de poder comunicarte con otro usuario en una conversación privada, también se pueden establecer conversaciones con varios usuarios, donde sólo se puedan consultar noticias o información publicadas por el administrador del grupo o; por el contrario, interactuar con los demás miembros del grupo de forma libre.
- *Telegram* <sup>12</sup>: Aplicación de código abierto y gratuita, con unas funcionalidades muy similares a las de WhatsApp.

Sin embargo, a la hora de integrar bots, WhatsApp no incluye esta tecnología. Aunque Facebook Messenger cuenta con soporte para bots, Telegram es una de las plataformas más populares a la hora de integrar chatbots debido a su extensa documentación y su excelente interfaz y experiencia de usuario. Por este motivo, se ha optado por utilizar Telegram.

<sup>9</sup> El 85% de usuarios utiliza servicios de mensajería online como WhatsApp o Telegram, <https://www.cnmc.es/node/375414>

<sup>10</sup> Facebook Messenger, <https://www.facebook.com/messenger/>

<sup>11</sup> WhatsApp Messenger, <https://www.whatsapp.com>

<sup>12</sup> Telegram Messenger, <https://telegram.org>

### 2.2.2 Herramientas de desarrollo de chatbots con motores IA

La funcionalidad principal de los motores de IA para el procesamiento del lenguaje formal es reconocer y clasificar la interacción del usuario. A continuación, se describen las herramientas más representativas actualmente para construir chatbots:

- *DialogFlow*<sup>13</sup>: Framework creado por Google que permite la creación y gestión de chatbots. Permite el procesamiento del lenguaje general (PNL), reconocimiento de voz, soporte tanto para intent como entities y especificación de respuestas. Es compatible con un gran número de idiomas y permite una integración con aplicaciones de mensajería amplia y sencilla.
- *Lex*<sup>14</sup>: Framework creado por Amazon que permite la creación y gestión de chatbots. Muchas características son similares a DialogFlow, sin embargo, sólo es compatible en inglés.
- *Watson*<sup>15</sup>: Framework creado por IBM que permite la creación y gestión de chatbots. Con características similares a DialogFlow, permite incluir expresiones regulares. Los idiomas compatibles y la integración con aplicaciones de mensajería es menor.
- *LUIS*<sup>16</sup>: Framework creado por Microsoft que permite la creación y gestión de chatbots. No maneja ni contextos ni respuestas, solo permite identificar la interacción del usuario y obtener la información relevante del mensaje.

Tras analizar todas las tecnologías anteriormente descritas, se opta finalmente por utilizar DialogFlow, ya que soporta más idiomas, y cuenta con un mayor número de aplicaciones de mensajería con las que puede integrarse.

DialogFlow define una serie de componentes que permiten manejar aspectos de la conversación:

- *Entities*: Conjuntos de palabras claves que el agente es capaz de identificar y clasificar.
- *Intents (Interacciones)*: Conjuntos de frases de entrenamiento que recoge la intención del usuario. Cuando un usuario escriba una frase, la herramienta la compara con los diferentes conjuntos de frases de entrenamiento hasta que encuentre el intent al que pertenece dicha frase. Los intents integran una serie de parámetros que ayudan a la hora de definir cada uno de ellos:
  - *Contexto*: Es similar al contexto del lenguaje natural; representa la información que rodea a la conversación. Mediante estos, se puede controlar el flujo de esta.
  - *Frases de entrenamiento*: Frases de ejemplo que el usuario al escribir, DialogFlow hace coincidencia con el intent.
  - *Acciones y parámetros*: Datos estructurados que se pueden usar para realizar alguna lógica o generar respuestas.
  - *Respuesta*: Mensaje de texto o imagen que DialogFlow devuelve como resultado del intent que se ha activado. Esta respuesta puede ser fija o devuelta por un servidor web externo.
- *Fulfillment*: Capacidad para comunicar un cierto intent con un servicio web. Esta comunicación se realiza a través de mensajes en formato Webhook Request y Webhook Response (ver Anexo A).

---

<sup>13</sup> DialogFlow, <https://cloud.google.com/dialogflow/docs>

<sup>14</sup> Lex, <https://docs.aws.amazon.com/lex/>

<sup>15</sup> Watson, <https://www.ibm.com/es-es/watson>

<sup>16</sup> LUIS, <https://docs.microsoft.com/es-es/azure/cognitive-services/luis/what-is-luis>

Se definen varios intents con diferentes frases de entrenamiento para cada interacción del usuario, y diferentes entities para extraer la información que el usuario ha especificado en el mensaje. Esto crea un motor IA capaz de identificar los intents y entities, y construir una salida según una frase predefinida en un intent, o un servicio web conectado con DialogFlow.

### 2.2.3 Servicio web

Como framework para el desarrollo del servidor web, se ha optado por Spring Boot <sup>17</sup> frente a JAX-WS.

Esta decisión viene dada debido a diferentes factores: Spring Boot es uno de los frameworks más utilizados actualmente para desarrollar servicios webs en lenguaje Java, además de contar con una documentación muy detallada y de gran calidad. Permite construir muchos componentes de manera automática, o con APIs ya desarrolladas, ahorrando tiempo en el desarrollo de estas funcionalidades. Además, tiene una configuración muy sencilla para desplegarse en Heroku <sup>18</sup>. Y respecto a la conexión con la base de datos, hemos decidido utilizar la API de persistencia JPA. La principal ventaja de esta API es la sencillez de mantenimiento y desarrollo, y la eliminación de errores en tiempo de ejecución (se detecta al desplegar la aplicación). Además, simplifica la configuración de Maven.

### 2.2.4 Base de datos relacional

El sistema de gestión de bases de datos relacionales permite almacenar y gestionar los datos de la base de datos relacional. A continuación, se describen los más importantes:

- *PostgreSQL* <sup>19</sup>: Motor de base de datos de código libre más potente y robusto. Cuenta con innumerables ventajas, como su gran escalabilidad, estabilidad y seguridad. Además, implementa todas las funcionalidades del estándar SQL.
- *SQLite* <sup>20</sup>: Software gratuito y de código abierto que, a diferencia del resto de motores de bases de datos relacionales, accede a la base de datos directamente, eliminando complejidad y configuración en los programas que lo integran.
- *MySQL* <sup>21</sup>: Motor más extendido. Brinda velocidad y confiabilidad, a expensas de una total adherencia al estándar SQL.

Al desplegarse en Heroku el servidor web, se ha optado por incorporar una base de datos PostgreSQL alojada en Heroku Postgres <sup>22</sup>, de esta manera, su configuración e integración resulta muy sencilla.

---

<sup>17</sup> Spring Boot, <https://spring.io/projects/spring-boot>

<sup>18</sup> Heroku, <https://devcenter.heroku.com/articles/getting-started-with-java>

<sup>19</sup> PostgreSQL, <https://www.postgresql.org/docs/>

<sup>20</sup> SQLite, <https://www.sqlite.org/index.html>

<sup>21</sup> MySQL, <https://www.mysql.com/>

<sup>22</sup> Heroku Postgres, <https://devcenter.heroku.com/articles/heroku-postgresql>



## 3 Diseño

Tras concluir el estudio de los frameworks de los cuatro componentes que integran el sistema conversacional, en este apartado se profundiza con el diseño de los tres componentes a desarrollar.

El diseño se ha estructurado en cuatro apartados:

1. Se define el esquema general de interacción.
2. Se detalla el diseño de la parte de la herramienta DialogFlow, explicando en detalle las soluciones que se han diseñado para cumplir todas las exigencias de la aplicación.
3. Se detalla el diseño del servidor web, y se profundiza en la arquitectura que integra y su paquetización.
4. Se detalla el diseño de la base de datos relacional PostgreSQL.

### 3.1 Esquema general interacción

El objetivo del chatbot es establecer una comunicación con el servicio web a través de DialogFlow, y devolver al usuario una respuesta. El flujo de esta interacción, representado en la figura 3-1, es el siguiente:

- Un usuario escribe un mensaje a través de la plataforma de mensajería, en este caso Telegram.
- La herramienta DialogFlow procesa el mensaje en lenguaje natural escrito por el usuario, y clasifica el texto, extrayendo el intent y la información que el usuario ha solicitado.
- El servicio web, alojado en Heroku, realiza la extracción de la información que DialogFlow le ha enviado en formato Webhook Request, la analiza y construye la respuesta según los valores analizados y tras consultar la base de datos relacional PostgreSQL, si esta ha sido necesaria.
- DialogFlow recibe la respuesta en formato Webhook Response del servicio web, y construye la salida que se envía a Telegram.
- El usuario visualiza la información que había consultado.

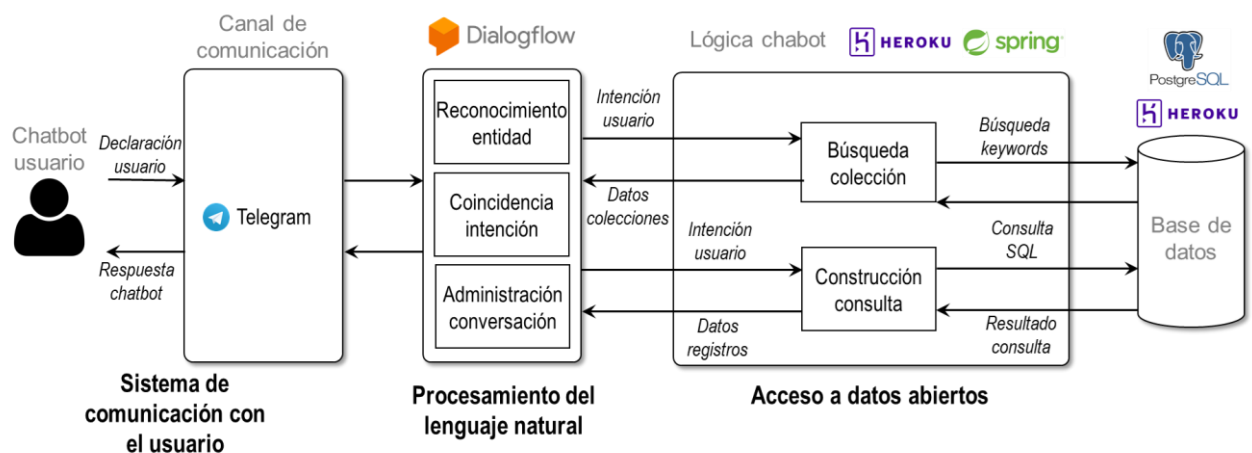


Figura 3-1: Esquema general de los componentes

Esta interacción tiene como finalidad recuperar los datos abiertos de diferentes colecciones del Ayuntamiento de Madrid <sup>23</sup>. Se han optado por incorporar seis colecciones diferentes:

- Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid, del año 2018.
- Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid, del año 2019.
- Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid, del año 2020.
- Calendario laboral.
- Decide Madrid.
- Presupuesto General. Año 2020.

Con el objetivo de poder ir integrando y añadiendo nuevas colecciones en el futuro, se realiza un diseño con especial énfasis en la flexibilidad, escalabilidad y mantenimiento del código para futuras integraciones, actualizaciones y modificaciones.

### **3.2 Diseño de la interacción**

Como paso previo al inicio del desarrollo del diseño, se estudia la documentación que Google tiene disponible para trabajar con la plataforma DialogFlow, además de consultar ejemplos y trabajos previos sobre una aplicación web, y como gestionar cada componente.

Tras dicho estudio, se extrajeron varias conclusiones respecto al diseño de la aplicación:

- Todos los intents que forman parte de la aplicación están relacionados, por lo que solo es necesario un único agente virtual que maneje las conversaciones en DialogFlow, llamado ChatbotOpenData.
- Diseñar nueve intents, que albergan todos los posibles casos en la conversación entre el usuario y el chatbot. Al margen de estos nueve, DialogFlow tiene un intent por defecto, Default Fallback Intent, que se activa al no reconocer el mensaje del usuario en ninguna de las frases de entrenamiento de los intents.
- No diseñar entities para identificar y extraer datos de los mensajes del usuario ya que, al haber múltiples colecciones y multitud de distintos keywords, es inviable el mantenimiento de dicha cantidad de entities. Solo se ha añadido un entity para el intent Satisfactory Response, e identificar si el usuario ha respondido de manera afirmativa o negativa a la cuestión que se le plantea acerca de la satisfacción del resultado obtenido.
- El servicio web es desarrollado en Spring Boot, con una Arquitectura de Tres Capas, donde a través de un controlador y una única ruta, se implementa toda la funcionalidad de comunicación entre el usuario y el servidor. Adicionalmente, se añaden otras rutas para poder depurar y mantener el servidor.
- Para almacenar todos los datos se utiliza una base de datos relacional PostgreSQL.
- Con el objetivo de poder realizar análisis de las interacciones de los usuarios, se añade en base de datos una tabla llamada Log para poder tener acceso a toda la información, para su posterior análisis y mejora del servidor.

A continuación, se detallan los diseños de los tres componentes que integran la aplicación.

---

<sup>23</sup> Catálogo datos abiertos Ayuntamiento Madrid, <https://datos.madrid.es/portal/site/egob/>



### 3.2.1 DialogFlow

Se ha valorado diseñar varios agentes que gestionen cada colección y su funcionalidad de manera independiente, y mantener sus entidades propias para cada campo.

Independientemente, otro agente integra todas las funcionalidades comunes a todas las colecciones. Esta solución es más costosa de crear y mantener, además de que en caso de uso normal del chatbot, se emplearían varios agentes en una misma conversación y mucha parte del código e implementación en DialogFlow y en el servidor estaría duplicado.

Sin embargo, tras analizar las necesidades de la aplicación, se decide integrar toda la funcionalidad de DialogFlow en un único agente que gestione todas las interacciones, y que establezca la comunicación con un único servidor. La ventaja principal de esta solución es la sencillez del diseño, ya que únicamente con nueve intents, se gestionan todas las interacciones.

Como se comenta en las conclusiones anteriores, se adopta un diseño donde DialogFlow no identifique las keywords relacionadas con ningún aspecto ni características de las colecciones en particular, salvo en el caso del intent Satisfactory Response.

#### Entities

Aunque en el comienzo del proyecto se intenta establecer varias entidades, las cuales sean identificadas, clasificadas y enviadas al servidor, sin necesidad de realizar ningún tratamiento del texto en este componente, tras ir integrando diferentes colecciones, y viendo la cantidad de datos y palabras distintas, se concluye que sería mejor que el servidor se encargue de esta tarea.

Al añadir una nueva colección, se debería realizar un análisis de la colección completa, extrayendo y clasificando cada dato, para crear varios conjuntos de keywords, y así, DialogFlow pudiera identificarlas cuando el usuario las introdujese. Una tarea muy compleja y costosa, que además dificultaría el diseño sencillo y claro que se estaba intentando realizar, donde un único agente de DialogFlow, con una cantidad reducida de intents, pudiera gestionar todas las interacciones de un usuario con el chatbot, sin necesidad de distinciones según colección o acción.

Las entidades utilizadas son:

- *Number*: Entité propia del sistema, de tipo entero, utilizada para identificar el número del identificador de la colección seleccionada por el usuario a consultar en el intent Select Collection.
- *Any*: Entité propia del sistema, sin tipo, utilizada para cualquier parámetro que contenga información introducida por el usuario. Es utilizada en los intents:
  - *Filter Keywords Collections*: Identifica las keywords introducidas que se usarán como filtro para las colecciones.
  - *Select Without Where*: Identifica los campos que se van a consultar, agrupar y ordenar; también se extrae el número de registros a devolver. Todos estos valores son de tipo entero, sin embargo, se ha optado por esta entité para poder añadir que el usuario introduzca texto para los casos en los que se quiera introducir los valores “Todo” (consultar todos los identificadores) o “No” (Si no se desea agrupar, ordenar o limitar el número final de registros).
  - *Select With Where*: Identifica los campos que se van a consultar, filtrar, agrupar y ordenar; también se extrae el número de registros a devolver.

Todos estos valores son de tipo entero, sin embargo, se ha optado por esta entitie para poder añadir que el usuario introduzca texto para los casos en los que se quiera introducir los valores “Todo” (consultar todos los identificadores) o “No” (Si no se desea agrupar, ordenar o limitar el número final de registros). A la hora de filtrar, además de introducir el identificador numérico, también se indica el tipo y el valor del filtro.

- *Satisfactory*: Entitie no propia del sistema, es decir, creada para registrar las respuestas afirmativas o negativas respecto a una cuestión. Es utilizada en el intent: Satisfactory Response.

## Intents

Como se ha descrito anteriormente, se han diseñado nueve intents (ver tabla 3-1) que albergan todas las intenciones que un usuario puede introducir:

Nombre	Descripción
<b>Welcome</b>	Se devuelve un mensaje de bienvenida o saludo al usuario.
<b>Help</b>	Se devuelve un mensaje con las posibles frases e interacciones que el chatbot puede responder.
<b>Get Collections</b>	Se devuelve un mensaje con todas las colecciones que el usuario puede consultar a través del chatbot.
<b>Filter Keywords Collections</b>	Se devuelve un mensaje con todas las colecciones que cumplan el filtro introducido por el usuario. Este filtro será una o varias palabras claves, que han sido definidas en un fichero que identifica cada colección.
<b>Select Collection</b>	Se devuelve un mensaje indicando si la colección elegida por el usuario para consultar es correcta o no. En caso de ser, se actualiza la base de datos para que el usuario pueda consultarla; en caso contrario, se comunica al usuario indicándole las colecciones disponibles a consultar.
<b>Get Collection Fields</b>	Se devuelve un mensaje listando todos los campos que componen la colección previamente indicada.
<b>Select Without Where</b>	Se devuelve un mensaje con los campos que el usuario ha solicitado. En este caso, no se aplica ningún tipo de filtro. Por último, se pregunta al usuario por la satisfacción de la información devuelta.
<b>Select With Where</b>	Se devuelve un mensaje con los campos que el usuario ha solicitado. En este caso, se pueden aplicar varios filtros. Por último, se pregunta al usuario por la satisfacción de la información devuelta.
<b>Satisfactory Response</b>	Se consulta al usuario si la respuesta de los intents Select Without Where o Select With Where ha sido satisfactoria o no.

**Tabla 3-1: Descripción intents**

Para cada uno de los nueve intents anteriormente descritos, se definen una serie de expresiones almacenadas como frases de entrenamiento por el agente de DialogFlow. Si la frase introducida no se corresponde a ninguna de las frases de entrenamiento del agente (ver tabla 3-2), se activa el intent Default Fallback Intent.

Nombre	Frases de entrenamiento
<b>Welcome</b>	<ul style="list-style-type: none"> <li>- Buenos días</li> <li>- Hola</li> <li>- Hey</li> <li>- Saludos</li> </ul>
<b>Help</b>	<ul style="list-style-type: none"> <li>- Necesito ayuda</li> <li>- Ayuda</li> </ul>
<b>Get Collections</b>	<ul style="list-style-type: none"> <li>- Muéstrame todas las colecciones</li> <li>- Colecciones disponibles</li> <li>- Muéstrame la lista de las colecciones a las que tengo acceso</li> <li>- ¿A qué colecciones tengo acceso?</li> <li>- ¿Puedes mostrarme todas las colecciones del sistema?</li> <li>- ¿Puedes mostrarme todas las colecciones?</li> </ul>
<b>Filter Keywords Collections</b>	<ul style="list-style-type: none"> <li>- Filtrar colecciones</li> <li>- Buscar colección</li> <li>- Consultar colecciones</li> <li>- No encuentro la colección</li> <li>- Buscar colección por keywords</li> </ul>
<b>Select Collection</b>	<ul style="list-style-type: none"> <li>- La número 1</li> <li>- 1</li> <li>- Colección 1</li> <li>- Quiero consultar la colección 1</li> <li>- Querría ver la colección 1</li> <li>- Muéstrame la colección 1</li> </ul>
<b>Get Collection Fields</b>	<ul style="list-style-type: none"> <li>- ¿Puedes mostrarme todos los campos de la colección?</li> <li>- Columnas disponibles</li> <li>- ¿Puedes mostrarme todas las columnas de la colección?</li> <li>- Muéstrame la lista de los datos a las que tengo acceso</li> <li>- Campos disponibles</li> </ul>
<b>Select Without Where</b>	<ul style="list-style-type: none"> <li>- Consulta sin filtrado</li> <li>- Consulta sin filtrar</li> <li>- Quiero realizar una consulta sin filtrado</li> </ul>
<b>Select With Where</b>	<ul style="list-style-type: none"> <li>- Consulta con filtro</li> <li>- Consulta filtrando</li> <li>- Quiero realizar una consulta con filtrado</li> </ul>
<b>Satisfactory Response</b>	<ul style="list-style-type: none"> <li>- No (Entitie Satisfactory)</li> <li>- Mucho (Entitie Satisfactory)</li> </ul>

**Tabla 3-2: Frases de entrenamiento de los intents**

A continuación, se establecen las acciones y las respuestas que el agente debe realizar una vez ha identificado el intent correspondiente. En todos los intent se ha llevado a cabo un diseño similar, donde se establece una conexión con el servidor, el cual envía una respuesta con el resultado de la consulta. Si esta conexión falla, DialogFlow tiene una respuesta predeterminada con el mensaje: “*Error al conectar con el servidor*”.

### 3.2.2 Servidor web

El servidor web se ha desarrollado con el framework Spring Boot, y desplegado en Heroku. Tras analizar su respectiva documentación, se ha llevado a cabo un estudio sobre las posibles arquitecturas que podría tener el servidor <sup>24</sup>. Al estar compuesto de tres componentes: comunicación con el cliente (Vista), la capa de negocio y la capa de acceso a datos; las arquitecturas más extendidas son:

- *Arquitectura Modelo Vista Controlador (MCV)*: Es una arquitectura triangular en la que hay una relación entre todos sus componentes.
- *Arquitectura a Tres Capas*: Es una arquitectura lineal, donde no hay comunicación directa entre todas las capas. La capa de presentación solo se puede comunicar con la capa intermedia o de negocio, y esta a su vez con la de datos. La capa de datos y la capa de presentación no pueden comunicarse.

Dado que se quiere tener una independencia de la capa de acceso a datos respecto a la de cliente, se opta por la Arquitectura de Tres Capas. Además, esta arquitectura permite una mayor flexibilidad a la hora de añadir más componentes, y dado que el número de colecciones va a ser incrementado, es otra ventaja respecto a la arquitectura MVC.

A continuación, se detallan las diferentes capas y sus características:

- *Capa de presentación y controlador*: Se encarga de la comunicación entre el servidor y el usuario y viceversa. Esta capa presenta la información al usuario, y obtiene la información de este.  
En esta aplicación, no hay una interfaz de usuario, sino que la comunicación es con DialogFlow; por lo que se encarga de recibir y obtener la información de la solicitud Webhook Request, y construir y enviar la respuesta Webhook Response.
- *Capa de negocio*: Es la capa donde se recibe la información ya obtenida del usuario y se procesa. Se encuentra toda la lógica de la aplicación. Se comunica con ambas capas.
- *Capa de acceso a datos*: Esta capa almacena los datos del sistema y del usuario. Su función es almacenar y devolver los datos a la capa superior, la capa de negocio.

La aplicación se ha dividido en tres módulos diferentes, correspondientes a las tres capas de la arquitectura:

- *Controller*: Se ha diseñado un controlador basado en API REST denominado ChatbotController, con las diferentes rutas necesarias para el correcto funcionamiento de la aplicación (ver 4.2.1). Estas rutas se detallan a continuación:
  - */ (home)*: Ruta donde está desplegada la funcionalidad completa del servidor. Debido a que va a recibir una información en formato JSON, el método es POST.
  - */filesResults/txt/{fileName}*: Ruta que devuelve el fichero de texto con el nombre fileName. Esta será accedida cuando el usuario pulse el botón de descarga del fichero de texto tras recibir el resultado de una consulta.

---

<sup>24</sup> Arquitectura de tres capas,

[https://www.ibm.com/support/knowledgecenter/es/SS7K4U\\_9.0.5/com.ibm.websphere.zseries.doc/ae/covr\\_3-tier.html](https://www.ibm.com/support/knowledgecenter/es/SS7K4U_9.0.5/com.ibm.websphere.zseries.doc/ae/covr_3-tier.html)

- */filesResults/excel/{fileName}*: Ruta que devuelve el fichero en formato Excel con el nombre fileName. Esta será accedida cuando el usuario pulse el botón de descarga del fichero Excel tras recibir el resultado de la consulta.
- *Service*: Se ha diseñado la lógica de negocio general en la interfaz IChatbotService, la cual implementa toda la lógica (ver 4.2.2). Adicionalmente, cada tabla de la base de datos tiene una interfaz para cubrir su lógica de negocio propia.
- *Model y Repository*: Por último y correspondiente a la última capa, se encuentran los modelos de cada tabla de la base de datos, así como la lógica para el acceso a su información a través de la API JPA y el driver JDBC (ver 4.2.3).

### 3.2.3 Base de datos

La base de datos PostgreSQL está alojada en Heroku Postgres, un servicio de base de datos SQL administrado por Heroku.

Se definen los requisitos de la base de datos:

- Los usuarios son únicos, con un identificador y nombre.
- Las colecciones son únicas, con un identificador, nombre y el nombre de la tabla donde se almacena la información de esta.
- La keyword tiene un nombre.
- Una keyword puede tener varias colecciones.
- Una colección puede tener varias keywords.
- Un usuario puede consultar una colección.
- Una colección puede ser consultada por varios usuarios a la vez.
- El log del servidor tiene que albergar el intent correspondiente a la acción, el estado (si se ha producido un error o no) y la fecha a la que se realiza.
- El log debe guardar el identificador de la colección que el usuario tiene seleccionada para dicho intent.
- El log en los intent Select Without Where y Select With Where, guarda el texto y la sentencia en SQL de la consulta hecha por el usuario. Además, se guarda si la consulta ha sido satisfactoria o no (Satisfactory Response).
- Una colección puede estar registrada varias veces en el log.
- Un usuario puede haber sido registrado en el log varias veces.
- Las colecciones introducidas deben albergar el contenido de los datos abiertos.
- Se añaden seis colecciones, las cuales van a estar contenidas en cuatro tablas.
- Estas tablas van a tener un identificador único autogenerado a través de una secuencia.
- Retribución de Madrid y Presupuesto General de Madrid tienen los campos comunes todos los años, así que se añade un campo para identificar el año.
- Las colecciones Retribución Madrid 2018, 2019 y 2020 se almacenan en la misma tabla.

Ante estos requisitos, se desarrolla la siguiente base de datos PostgreSQL, con ocho tablas y seis secuencias [3]. A continuación, se detallan sus diseños.

### Tabla Usuario

Esta tabla almacena la información de los usuarios que hagan uso de la aplicación. Los usuarios se detallan como una entidad de tres atributos: un identificador, el nombre del usuario y el identificador de la colección seleccionada por el usuario.

Nombre	Tipo	Descripción	Restricciones
id	bigint	Identifica el usuario. Se extrae de Telegram	Clave primaria
nombre	text	Nombre del usuario. Se extrae de Telegram	-
id_coleccion	integer	Identificador de la colección elegida por el usuario	Clave foránea de la tabla Colección (id)

Tabla 3-3: Detalle tabla Usuario

### Tabla Colección

Esta tabla almacena la información de las colecciones que el sistema pueda consultar, en este caso seis.

Las colecciones se detallan como una entidad de tres atributos: un identificador, el nombre de la colección y el nombre de la tabla de la base de datos a la que corresponde la colección.

Nombre	Tipo	Descripción	Restricciones
id	integer	Identifica la colección	Clave primaria.
nombre	text	Nombre de la colección	-
tabla	text	Nombre de la tabla en base de datos	-

Tabla 3-4: Detalle tabla Colección

### Tabla Keyword

Esta tabla almacena las palabras claves que identifican a cada colección, para que el usuario pueda buscarlas a través de esos términos.

Las keywords se detallan como una entidad de tres atributos: un identificador, el nombre de la keyword y el identificador de la colección a la que corresponde la keyword.

Nombre	Tipo	Descripción	Restricciones
id	bigint	Identifica cada relación entre la colección y la keyword	Clave primaria. Valor autogenerado de la secuencia sec_id_keyword
keyword	text	Palabra clave	-
id_coleccion	integer	Identificador de la colección a la que hace referencia la keyword	Clave foránea de la tabla Colección (id)

Tabla 3-5: Detalle tabla Keyword

### Tabla Log

Esta tabla almacena la información de las interacciones y acciones de los usuarios con el sistema.

Los logs se detallan como una entidad de nueve atributos: un identificador, el usuario que ha realizado la interacción, el identificador de la colección que se consulta o se ha seleccionado, la hora de la interacción, el intent, el estado de esta interacción, la consulta en formato texto, la consulta en formato SQL y si la consulta ha sido satisfactoria.

Nombre	Tipo	Descripción	Restricciones
<b>id</b>	bigint	Identifica cada log	Clave primaria. Valor autogenerado de la secuencia sec_id_log
<b>id_usuario</b>	bigint	Identifica el usuario que ha realizado la acción	Clave foránea de la tabla Usuario (id)
<b>id_coleccion</b>	integer	Identificador de la colección que se utiliza	Clave foránea de la tabla Colección (id)
<b>fecha</b>	timestamp	Fecha a la que se registró la acción	-
<b>intent</b>	text	Nombre del intent	-
<b>estado</b>	text	Estado de la interacción	-
<b>consulta</b>	text	Consulta realizada en formato texto	-
<b>consulta_sql</b>	text	Consulta realizada en formato SQL	-
<b>consulta_satisfactoria</b>	text	Campo que indica si la consulta ha sido satisfactoria	-

Tabla 3-6: Detalle tabla Log

### Tabla Calendario\_Madrid

Esta tabla almacena el calendario laboral de la ciudad de Madrid desde 2013 hasta 2021.

El calendario de Madrid se detalla como una entidad de seis atributos: un identificador y los cinco campos propios de la colección.

Nombre	Tipo	Descripción	Restricciones
<b>id</b>	bigint	Identifica cada día	Clave primaria. Valor autogenerado de secuencia sec_id_calendario_madrid
<b>dia</b>	text	Campo de la colección	-
<b>dia_semana</b>	text	Campo de la colección	-
<b>tipo</b>	text	Campo de la colección	-
<b>tipo_de_festivo</b>	text	Campo de la colección	-
<b>festividad</b>	text	Campo de la colección	-

Tabla 3-7: Detalle tabla Calendario\_Madrid

### Tabla Retribución\_Madrid

Esta tabla almacena las retribuciones de los años 2018, 2019 y 2020.

Las retribuciones de Madrid se detallan como una entidad de treinta y cinco atributos: un identificador, el año de la retribución y los treinta y tres campos propios de la colección.

Nombre	Tipo	Descripción	Restricciones
<b>id</b>	bigint	Identifica cada retribución	Clave primaria. Valor autogenerado de secuencia sec_id_retribucion_madrid
<b>año</b>	integer	Año de la retribución	-
<b>denominacion colectiva del puesto</b>	text	Campo de la colección	-
<b>unidad de adscripcion</b>	text	Campo de la colección	-
<b>numero de puesto</b>	integer	Campo de la colección	-
<b>denominacion individual del puesto</b>	text	Campo de la colección	-
<b>nombre y apellidos</b>	text	Campo de la colección	-
<b>fecha de acuerdo</b>	text	Campo de la colección	-
<b>numero de boam</b>	integer	Campo de la colección	-
<b>fecha de boam</b>	text	Campo de la colección	-
<b>codigo de cargo</b>	text	Campo de la colección	-
<b>regimen dedicacion en enero</b>	text	Campo de la colección	-
<b>retribucion bruta en enero</b>	real	Campo de la colección	-
<b>regimen dedicacion en febrero</b>	text	Campo de la colección	-
<b>retribucion bruta en febrero</b>	real	Campo de la colección	-
<b>regimen dedicacion en marzo</b>	text	Campo de la colección	-
<b>retribucion bruta en marzo</b>	real	Campo de la colección	-
<b>regimen dedicacion en abril</b>	text	Campo de la colección	-
<b>retribucion bruta en abril</b>	real	Campo de la colección	-
<b>regimen dedicacion en mayo</b>	text	Campo de la colección	-
<b>retribucion bruta en mayo</b>	real	Campo de la colección	-
<b>regimen dedicacion en junio</b>	text	Campo de la colección	-
<b>retribucion bruta en junio</b>	real	Campo de la colección	-
<b>regimen dedicacion en julio</b>	text	Campo de la colección	-
<b>retribucion bruta en julio</b>	real	Campo de la colección	-
<b>regimen dedicacion en agosto</b>	text	Campo de la colección	-
<b>retribucion bruta en agosto</b>	real	Campo de la colección	-
<b>regimen dedicacion en septiembre</b>	text	Campo de la colección	-
<b>retribucion bruta en septiembre</b>	real	Campo de la colección	-
<b>regimen dedicacion en octubre</b>	text	Campo de la colección	-
<b>retribucion bruta en octubre</b>	real	Campo de la colección	-
<b>regimen dedicacion en noviembre</b>	text	Campo de la colección	-
<b>retribucion bruta en noviembre</b>	real	Campo de la colección	-
<b>regimen dedicacion en diciembre</b>	text	Campo de la colección	-
<b>retribucion bruta en diciembre</b>	real	Campo de la colección	-

Tabla 3-8: Detalle tabla Retribución\_Madrid



### Tabla Decide\_Madrid

Las propuestas del portal Decide Madrid se detallan como una entidad de diez atributos: un identificador y los nueve campos propios de la colección.

Nombre	Tipo	Descripción	Restricciones
id	bigint	Identifica cada propuesta	Clave primaria. Valor autogenerado de secuencia sec_id_decide_madrid
identificador	integer	Campo de la colección	-
url	text	Campo de la colección	-
título	text	Campo de la colección	-
fecha	text	Campo de la colección	-
resumen	text	Campo de la colección	-
numero_apoyos	integer	Campo de la colección	-
numero_comentarios	integer	Campo de la colección	-
tema	text	Campo de la colección	-
distrito	text	Campo de la colección	-

Tabla 3-9: Detalle tabla Decide\_Madrid

### Tabla Presupuesto\_General\_Madrid

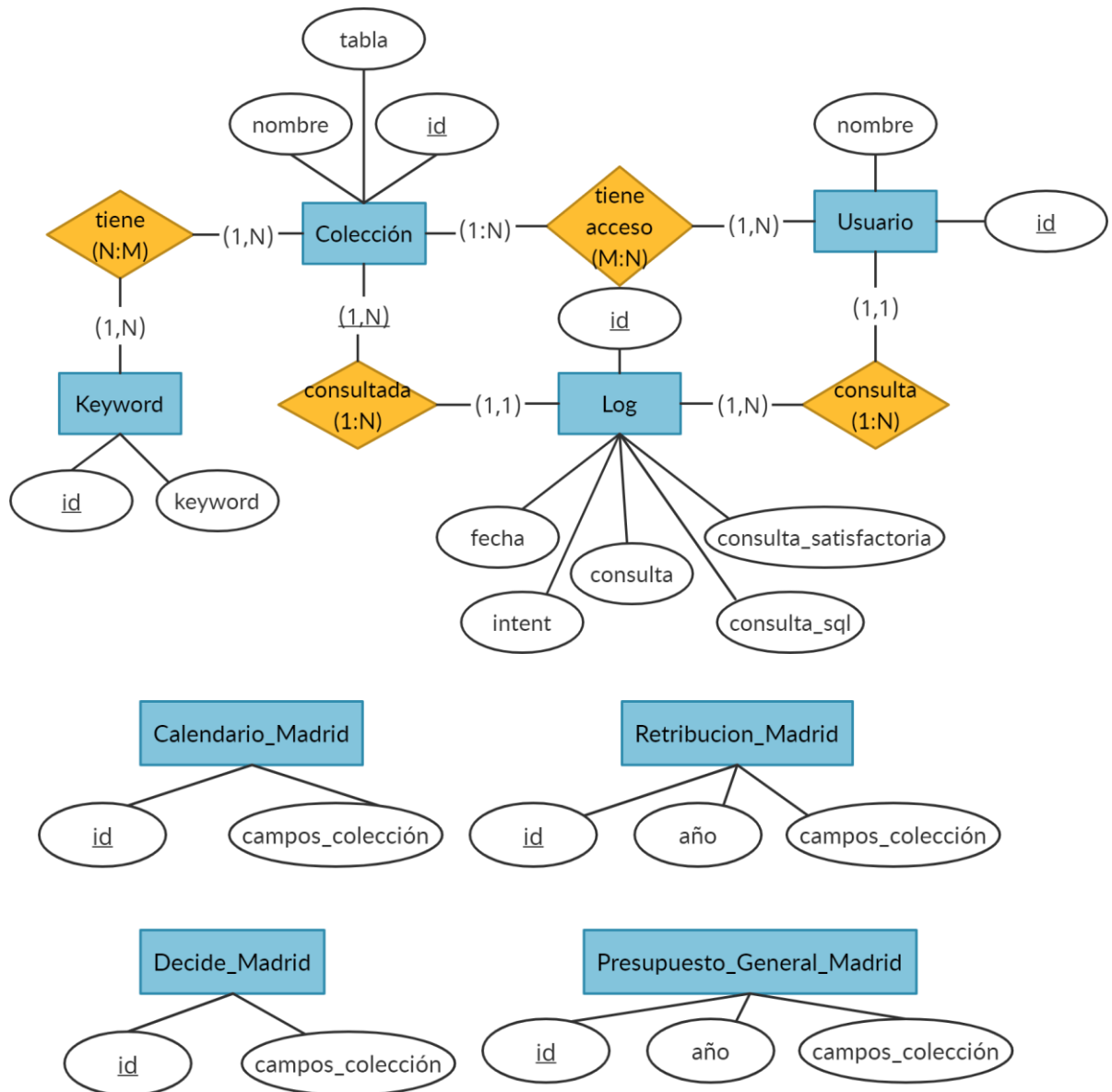
Los Presupuestos Generales de Madrid se detallan como una entidad de trece atributos: un identificador, el año del presupuesto y los once campos propios de la colección.

Nombre	Tipo	Descripción	Restricciones
id	bigint	Identifica cada presupuesto	Clave primaria. Valor autogenerado de secuencia sec_id_presupuesto_general_madrid
año	integer	Año del presupuesto	-
centro	text	Campo de la colección	-
descripcion_centro	text	Campo de la colección	-
seccion	integer	Campo de la colección	-
descripcion_seccion	text	Campo de la colección	-
programa	text	Campo de la colección	-
descripcion_programa	text	Campo de la colección	-
capitulo	integer	Campo de la colección	-
descripcion_capitulo	text	Campo de la colección	-
economico	text	Campo de la colección	-
relacion_de_partidas	text	Campo de la colección	-
presupuesto	real	Campo de la colección	-

Tabla 3-10: Detalle tabla Presupuesto\_General\_Madrid

### Diagrama Entidad Relación

Se adjunta en la figura 3-2 el diagrama ER de la base de datos completa. Se indican todas las entidades, atributos y relaciones de la base de datos.



**Figura 3-2: Diagrama ER de la base de datos**

## 4 Desarrollo

Tras describir el diseño de los tres componentes que se desarrollan de la aplicación, en este apartado se profundiza en el desarrollo de estos.

El desarrollo se ha estructurado en dos apartados:

1. Se detalla la estructura de flujo del sistema, explicando y analizando cada intent.
2. Se detalla el desarrollo del servidor web. Se profundiza en la funcionalidad de los tres bloques de la arquitectura, detallando los métodos que integran cada uno de ellos y su funcionalidad.

### 4.1 Estructura del flujo

Tras diseñar los nueve intents de la aplicación, se detalla cada uno individualmente, indicando los parámetros obtenidos del mensaje por parte de DialogFlow.

La figura 4-1 muestra el flujo de la conversación del chatbot [4]. Una conversación se compone de intenciones que representan diferentes necesidades de los usuarios (propósitos u objetivos). Una intención puede ser independiente del resto de las intenciones, o debe considerarse después de abordar otra intención.

En una conversación, cada intent se activa cuando el usuario escribe una frase que satisface cierto patrón de oración, que se define de antemano en DialogFlow (ver 3.2.1). Cuando una intención es activada, el chatbot llama a un servicio web externo para ejecutar ciertas funcionalidades. Los resultados de la rutina se devuelven al chatbot, que finalmente se presentan al usuario.

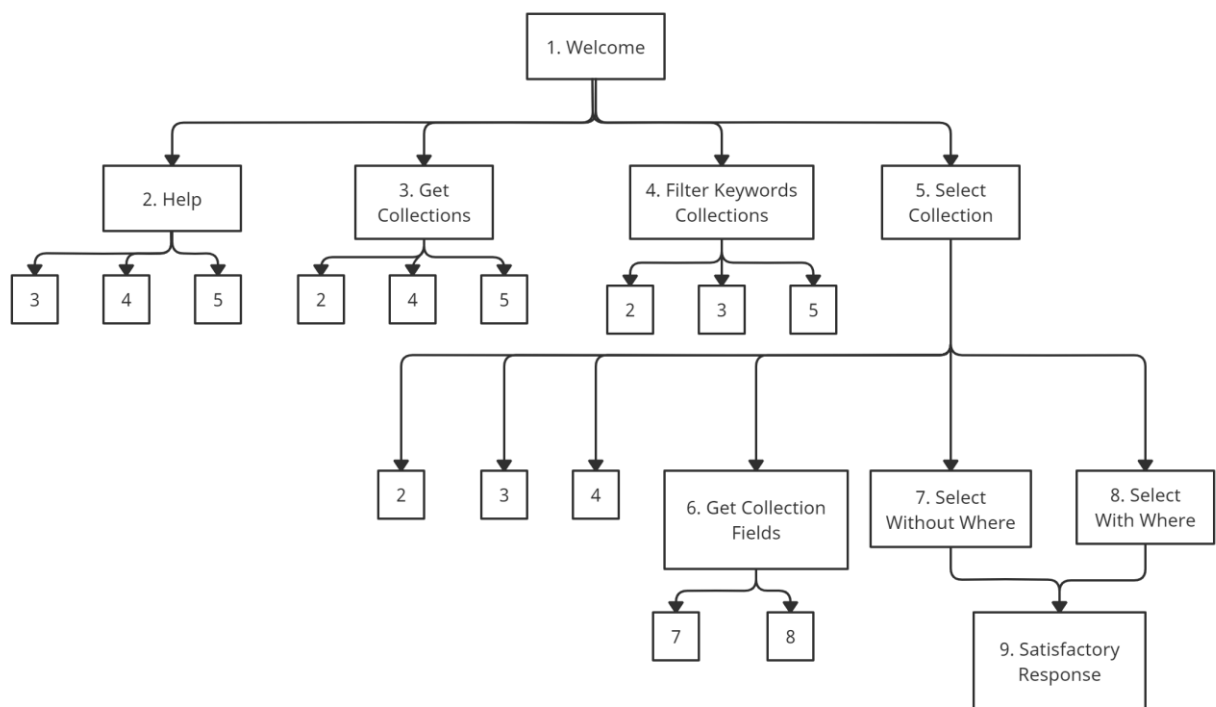


Figura 4-1: Diagrama de flujo de conversación

A continuación, describimos las intenciones de las conversaciones definidas, presentando algunos de sus patrones de frases. En el Anexo B se añade un ejemplo de una conversación completa.

#### **4.1.1 Intent Welcome**

Este intent se activa al comenzar la conversación por primera vez, o cuando el usuario escribe un mensaje como:

- Buenos días.
- Hola.

Este intent no utiliza ningún parámetro. DialogFlow establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe un mensaje con la respuesta de bienvenida.

#### **4.1.2 Intent Help**

Este intent se activa cuando el usuario escribe un mensaje como:

- Necesito ayuda.
- Ayuda

Este intent no utiliza ningún parámetro. DialogFlow establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe un mensaje con todas las posibles opciones a consultar, así como las frases de entrenamiento que habilitan cada intent.

#### **4.1.3 Intent Get Collections**

Este intent se activa cuando el usuario escribe un mensaje como:

- Muéstrame las colecciones disponibles.
- ¿A qué colecciones tengo acceso?

Este intent no utiliza ningún parámetro. DialogFlow establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe la lista de todas las colecciones que se pueden consultar a través del chatbot.

#### **4.1.4 Intent Filter Keywords Collections**

Este intent se activa cuando el usuario escribe un mensaje como:

- Filtrar colecciones.
- Buscar colecciones por keywords.

Tras introducir una de estas frases, DialogFlow devuelve la frase “*Enumere las keywords de la colección que quiere buscar*”, donde solicita enumerar las keywords del filtro. Por ejemplo:

- 2020, concejal
- 2018

DialogFlow obtiene el parámetro Keyword, de tipo Any, con el mensaje introducido por el usuario. Se establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe primero la consulta que ha realizado, las palabras claves a filtrar. Tras este texto, se devuelve el resultado de la consulta, listando todas las colecciones que cumplen los filtros introducidos.

#### **4.1.5 Intent Select Collection**

Este intent se activa cuando el usuario escribe un mensaje como (Supuesto el usuario quiere escoger la colección número 1):

- Colección 1.
- Querría ver la colección 1.

DialogFlow obtiene el parámetro id\_collection, de tipo Number, con el identificador introducido por el usuario. Se establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe un mensaje indicando si la elección se ha realizado de manera correcta.

#### **4.1.6 Intent Get Collection Fields**

Este intent se activa cuando el usuario escribe un mensaje como:

- ¿Puedes mostrarme todos los campos de la colección?
- ¿Puedes mostrarme todas las columnas de la colección?

Este intent no utiliza ningún parámetro. Se establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe una lista de los identificadores y sus respectivos campos de la colección que ha seleccionado.

Para los intents *Select Without Where* (ver 4.1.7) y *Select With Where* (ver 4.1.8), se ha intentado dotar al chatbot con una alta diversidad de posibles consultas por parte del usuario. La instrucción utilizada es SELECT, combinándola con múltiples cláusulas como GROUB BY, ORDER BY, WHERE O LIMIT.

A continuación, se especifica una serie de ejemplos de posibles consultas que se pueden construir, indicándola primero en lenguaje natural, y posteriormente en la tabla 4-1 en formato SQL, junto a los parámetros que extraería DialogFlow:

1. Muéstrame los registros.
2. Muéstrame atr5 y atr6.
3. Muéstrame la suma de atr2 agrupando los resultados por atr5 y atr6.
4. Muéstrame los diez atr5 y atr6, ordenándolos de manera ascendente por atr3.

5. Muéstrame las tres sumas de atr2 agrupando los resultados por atr5 y atr6, ordenándolos de manera descendente por atr3.
6. Muéstrame los registros cuando el atr5 sea Díaz y atr7 sea mayor que 500.
7. Muéstrame las tres sumas de atr2 agrupando los resultados por atr5 y atr6, ordenándolos de manera descendente por atr3 cuando el atr5 sea Díaz y atr7 sea mayor que 500.

Parámetros	Consulta SQL
<b>1. id_field_select</b> <b>2. field_where</b> <b>3. id_field_groupby</b> <b>4. id_field_orderby</b> <b>5. number_limit</b>	
<b>1. Todo</b> <b>2. (No disponible)</b> <b>3. No</b> <b>4. No</b> <b>5. No</b>	SELECT * FROM colección
<b>1. 5, 6</b> <b>2. (No disponible)</b> <b>3. No</b> <b>4. No</b> <b>5. No</b>	SELECT atr5, atr6 FROM colección
<b>1. 5, 6</b> <b>2. (No disponible)</b> <b>3. SUM 2</b> <b>4. No</b> <b>5. No</b>	SELECT atr5, atr6, SUM(atr2) FROM colección GROUP BY atr5, atr6
<b>1. 5, 6</b> <b>2. (No disponible)</b> <b>3. No</b> <b>4. 3 ASC</b> <b>5. 10</b>	SELECT atr5, atr6 FROM colección ORDER BY atr3 ASC LIMIT 10
<b>1. 5, 6</b> <b>2. (No disponible)</b> <b>3. MIN 2</b> <b>4. 3 DESC</b> <b>5. 3</b>	SELECT atr5, atr6, SUM(atr2) FROM colección GROUP BY atr5, atr6 ORDER BY atr3 DESC LIMIT 3
<b>1. Todo</b> <b>2. 5 = Díaz, 7 &gt; 500</b> <b>3. No</b> <b>4. No</b> <b>5. No</b>	SELECT * FROM colección WHERE atr5 LIKE '%DIAZ%' AND atr7 > 500
<b>1. 5, 6</b> <b>2. 5 = Díaz, 7 &gt; 500</b> <b>3. MAX 3</b> <b>4. 3 DESC</b> <b>5. 10</b>	SELECT atr5, atr6, SUM(atr3) FROM colección WHERE atr5 LIKE '%DIAZ%' AND atr7 > 500 GROUP BY atr5, atr6 ORDER BY 3 DESC LIMIT 3

**Tabla 4-1: Tipos de consultas soportadas**

#### 4.1.7 Intent Select Without Where

Este intent se activa cuando el usuario escribe un mensaje como:

- Consulta sin filtro.
- Quiero realizar una consulta sin filtrado.
- Consulta sin filtrar

Tras introducir una de estas frases, DialogFlow devuelve la frase *“¿Qué campos quiere consultar? Introduzca los identificadores de los campos. Introduzca "Todo" si quiere consultar todos los campos”*, donde se solicita enumerar los campos que se quieren consultar. Estos identificadores se pueden consultar activando el intent anterior. Por ejemplo:

- Todo.
- 4, 5, 10.
- 5

DialogFlow obtiene el parámetro *id\_field\_select*, de tipo Any, con los identificadores de los campos. DialogFlow devuelve la frase *“¿Qué funciones de agregación quiere aplicar? Introduzca la función (SUM, MAX, MIN, AVG, COUNT) y el identificador del campo. Por ejemplo: "sum 5". Introduzca "No" si no quiere agrupar la consulta”*, donde se solicita la función de agregación a utilizar tras agrupar los campos. Por ejemplo:

- No.
- MAX 4.
- min 3.

DialogFlow obtiene el parámetro *id\_field\_groupby*, de tipo Any con la función de agrupación y el campo. DialogFlow devuelve la frase *“¿Por qué campo quiere ordenarla? Introduzca el identificador y el orden (ASC, DESC). Por ejemplo: "4 desc". Introduzca "No" si no quiere ordenar la consulta”*, donde se solicita por qué campos se quiere ordenar la salida. Por ejemplo:

- No.
- 5 asc.
- 3 DESC.

DialogFlow obtiene el parámetro *id\_field\_orderby*, de tipo Any, con el identificador del campo y el orden de ordenación. DialogFlow devuelve la frase *“¿Cuántos registros quiere que se muestren? Introduzca el número de registros a devolver. Introduzca "No" si quiere que se devuelvan todos los resultados.”*, donde se solicita el número de registros quiere que se devuelvan. Por ejemplo:

- No.
- 5.

DialogFlow obtiene el parámetro *number\_limit*, de tipo Any, con el número de registros a devolver. Se establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe primero qué consulta ha realizado: los campos a consultar, las agrupaciones, las ordenaciones y si hay límite de registros a devolver. Tras este texto, se devuelve el resultado de la consulta, aunque si el resultado es muy extenso, no podrá

observarse de forma preliminar. A continuación, se pueden observar dos botones, donde el usuario puede descargar el resultado de la consulta en formato de texto y en formato de hoja de cálculo. Por último, se pregunta al usuario si la respuesta ha sido satisfactoria, habilitando el intent descrito en el apartado 3.1.9, Satisfactory Response.

#### 4.1.8 Intent Select With Where

Este intent se activa cuando el usuario escribe un mensaje como:

- Consulta con filtro.
- Quiero realizar una consulta con filtrado.
- Consulta filtrando

Tras introducir una de estas frases, DialogFlow realiza la misma lógica que en la sección anterior (ver 4.1.7), a excepción del valor del filtro. Tras el usuario escribir los identificadores de los campos a consultar, DialogFlow devuelve la frase *“¿Qué filtros quiere aplicar? Enumera los identificadores de los campos, añadiendo el tipo de filtro (" $<$ " menor / " $\leq$ " menor o igual / " $>$ " mayor / " $\geq$ " mayor o igual / " $=$ " igual / " $\neq$ " distinto) y su valor. Por ejemplo: "5 = Carmena, 33 > 5000".*, donde se solicitan todos los filtros que el usuario quiera aplicar. Por ejemplo:

- 5 = Carmena, 33 > 5000.
- 5  $\neq$  Carmena.

DialogFlow obtiene el parámetro *field\_where*, de tipo Any, con los identificadores de los campos, el tipo de filtro y el contenido a filtrar. DialogFlow continúa con la misma lógica, extrayendo los campos *id\_field\_groupby*, *id\_field\_orderby* y *number\_limit*; estableciendo comunicación con el servidor web y retornando la respuesta a la consulta planteada por el usuario.

#### 4.1.9 Intent Satisfactory Response

Este intent se activa tras devolver los resultados de los dos intents anteriormente indicados exclusivamente. La respuesta del usuario puede ser afirmativa o negativa.

DialogFlow obtiene el parámetro Satisfactory, de tipo Satisfactory, con la satisfacción del usuario. Se establece conexión con el servicio web enviando la petición en formato Webhook Request. El servidor web devuelve el mensaje final en formato Webhook Response.

El usuario recibe un mensaje indicando si la respuesta se ha registrado de manera correcta.

### 4.2 Arquitectura del servidor web

Los diferentes ficheros que forman el servidor han sido estructurados en diferentes directorios según su tipo y finalidad<sup>25</sup>. Los directorios que forman este proyecto son:

- *Src/main/java*: Carpeta principal del proyecto que contiene todo el código fuente Java del servidor. Almacena toda la funcionalidad del proyecto.
- *Src/main/resources*: Carpeta que contiene la configuración del servidor. Esta información se almacena en cuatro ficheros según su tipo de configuración:
  - o *Application.properties*: Fichero con la configuración de la base de datos y de la conexión entre el servidor y la base de datos.

---

<sup>25</sup> Java™ Platform, Standard Edition 8 API Specification, <https://docs.oracle.com/javase/8/docs/api/>



- *Message.properties*: Fichero con los mensajes finales que el servidor retorna a DialogFlow. Este fichero está pensado para que se puedan añadir diferentes idiomas según la preferencia de idioma del usuario.
- *MessageError.properties*: Fichero con los mensajes de error que el servidor retorna a DialogFlow. Este fichero está pensado para que se puedan añadir diferentes idiomas según la preferencia de idioma del usuario.
- *SentencesSQL.properties*: Fichero con las consultas SQL que el servidor va a necesitar para las consultas a la base de datos.
- *Data*: Carpeta donde se almacenan los ficheros con la información de las colecciones incluidas en la base de datos. Estos han sido descargados de la página de datos abiertos del Ayuntamiento de Madrid <sup>3</sup>. Además, existe otro directorio:
  - *FilesResults*: Carpeta que almacena todos los ficheros de texto y Excel generados por el servidor con el resultado de las consultas planteadas por los usuarios. Estos ficheros pueden ser solicitados para su descarga.

Como se describe anteriormente (ver 3.2.2), el servidor se ha estructurado en tres bloques, con la correspondiente funcionalidad de las tres capas de la arquitectura.

#### 4.2.1 Capa de presentación y controlador

Esta capa se encarga de la comunicación entre el servidor y el usuario y viceversa. Esta capa presenta la información al usuario, y obtiene la información de este. En esta aplicación, no hay una interfaz de usuario, sino que la comunicación es con DialogFlow; por lo que se encarga de recibir y obtener la información de la solicitud Webhook Request, y construir y enviar la respuesta Webhook Response. Su código fuente está en el paquete *es.uam.eps.tfg.ChatbotOpenData.controller*.

Para este proyecto, se han desarrollado siete controladores, con una finalidad diferente:

- *CalendarioMadridController.java*: Se encarga de gestionar la tabla Calendario\_Madrid. Esta tarea es de mantenimiento, y no será accesible a través del chatbot. Se permite poblar la base de datos y visualizar todos los datos de la tabla.
- *ChatbotController.java*: Se encarga de gestionar la petición de DialogFlow, y alberga la funcionalidad de la aplicación.
- *CollectionController.java*: Se encarga de gestionar la tabla Colección. Esta tarea es de mantenimiento, y no será accesible a través del chatbot. Se permite poblar la base de datos y visualizar todos los datos de la tabla.
- *DecideMadridController.java*: Se encarga de gestionar la tabla Decide\_Madrid. Esta tarea es de mantenimiento, y no será accesible a través del chatbot. Se permite poblar la base de datos y visualizar todos los datos de la tabla.
- *KeywordController.java*: Se encarga de gestionar la tabla Keyword. Esta tarea es de mantenimiento, y no será accesible a través del chatbot. Se permite poblar la base de datos y visualizar todos los datos de la tabla.
- *PresupuestoGeneralMadridController.java*: Se encarga de gestionar la tabla Presupuesto\_General\_Madrid. Esta tarea es de mantenimiento, y no será accesible a través del chatbot. Se permite poblar la base de datos y visualizar todos los datos de la tabla, tanto los años por separado como todos los años a la vez.

- *RetribucionMadridController.java*: Se encarga de gestionar la tabla Retribución\_Madrid. Esta tarea es de mantenimiento, y no será accesible a través del chatbot. Se permite poblar la base de datos y visualizar todos los datos de la tabla, tanto los años por separado como todos los años a la vez.

Intent	Parámetros
<b>Welcome</b>	- id_user: Identificador del usuario
<b>Help</b>	- id_user: Identificador del usuario
<b>Get Collections</b>	- id_user: Identificador del usuario
<b>Filter Keywords Collections</b>	- id_user: Identificador del usuario - keywords: Palabras claves que identifican a una colección
<b>Select Collection</b>	- id_user: Identificador del usuario - id_collection: Identificador de la colección elegida
<b>Get Collection Fields</b>	- id_user: Identificador del usuario - id_collection: Identificador de la colección elegida. Se extrae de base de datos, de la tabla Usuario
<b>Select Without Where</b>	- id_user: Identificador del usuario - id_collection: Identificador de la colección elegida. Se extrae de base de datos, de la tabla Usuario - id_field_select: Identificadores de los campos a consultar - id_field_groupby: Identificador y función de agregación tras agrupar la consulta - id_field_orderby: Identificador y orden por campo al ordenar la consulta - number_limit: Número de registros a devolver
<b>Select With Where</b>	- id_user: Identificador del usuario - id_collection: Identificador de la colección elegida. Se extrae de base de datos, de la tabla Usuario - id_field_select: Identificadores de los campos a consultar - where_field: Filtros de la consulta - id_field_groupby: Identificador y función de agregación tras agrupar la consulta - id_field_orderby: Identificador y orden por campo al ordenar la consulta - number_limit: Número de registros a devolver
<b>Satisfactory Response</b>	- id_user: Identificador del usuario - satisfactory: Satisfacción del usuario a la consulta devuelta

**Tabla 4-2: Relación Intents - Parámetros**

El controlador ChatbotController es el que se encarga de gestionar toda la funcionalidad de la aplicación:

1. Se recibe una cadena de texto con el formato Webhook Request.
2. Utilizando la API de DialogFlow y para mayor sencillez, se parsea la cadena a tipo GoogleCloudDialogFlowV2WebhookRequest.
3. Se extrae el intent que DialogFlow ha identificado.
4. Se extraen todos los parámetros necesarios para cada intent que se encuentran en el objeto obtenido en el punto 2. Esta relación entre parámetros e intents se detalla en la tabla 4.2.
5. Se realiza la lógica de negocio según el intent (ver 4.2.2).

6. Se construye la respuesta con el texto obtenido del paso 5, en formato Webhook Response, `GoogleCloudDialogFlowV2WebhookResponse`.
7. Se inserta el registro correspondiente a esta interacción en la tabla Log.
8. Se devuelve a DialogFlow este mensaje con el resultado correspondiente al intent.

Los demás controladores tienen tareas de mantenimiento. Entre esas tareas, se incluye añadir registros a la tabla correspondiente de un fichero CSV. El sistema lee el fichero, extrae el contenido, lo procesa y lo añade a la base de datos. Este proceso es fundamental para el correcto desarrollo del experimento llevado a cabo (ver 5.2).

#### 4.2.2 Capa de negocio

Es la capa donde se recibe la información ya obtenida del usuario y se procesa. Su código fuente está en el paquete: *es.uam.eps.tfg.ChatbotOpenData.service*.

Al igual que ocurre en la capa de presentación, se han desarrollado nueve interfaces para desarrollar de forma independiente cada una de las lógicas de negocio:

- *ICalendaroMadridService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Calendario\_Madrid e interactuar con la capa de acceso a datos.
- *IChatbotService.java*: Se encarga de albergar toda la lógica de negocio de la aplicación.
- *ICollectionService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Colección e interactuar con la capa de acceso a datos.
- *IDecideMadridService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Decide\_Madrid e interactuar con la capa de acceso a datos.
- *IKeywordService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Keyword e interactuar con la capa de acceso a datos.
- *ILogService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Log e interactuar con la capa de acceso a datos.
- *IPresupuestoGeneralMadridService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Presupuesto\_General\_Madrid e interactuar con la capa de acceso a datos.
- *IRetribucionMadridService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Retribución\_Madrid e interactuar con la capa de acceso a datos.
- *IUserService.java*: Se encarga de albergar toda la lógica de negocio de la tabla Usuario e interactuar con la capa de acceso a datos.

Las ocho interfaces que gestionan las diferentes tablas implementan las funciones básicas en base de datos, CRUD.

La interfaz *IChatbotService* alberga un método por cada intent de la aplicación, por lo tanto, hay nueve métodos:

- *Welcome()*: Busca en el fichero de texto descrito anteriormente, `Message.properties`, el texto correspondiente a este intent.
- *Help()*: Busca en el fichero de texto `Message.properties` el texto correspondiente a este intent.
- *GetCollections()*: A través de la lógica de negocio de la interfaz *ICollectionService*, extrae de base de datos todos los nombres de las colecciones ordenados de manera ascendente por el identificador. Busca también en el

fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto.

- *FilterKeywordCollections()*: A través de la lógica de negocio de la interfaz `IKeywordService`, extrae de base de datos todas las colecciones que cumplen los filtros del usuario. Busca también en el fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto.
- *SelectCollections()*: A través de la lógica de negocio de la interfaz `IUserService`, añade la colección indicada. Anteriormente se ha comprobado que dicha colección existe. Busca también en el fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto.
- *GetCollectionFields()*: A través de la lógica de negocio de la interfaz de cada colección, por ejemplo, `IRetribucionMadridService`, extrae los nombres de los campos, junto a su identificador. Busca también en el fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto.
- *SelectWithoutWhere()*: A través de la lógica de negocio de la interfaz de cada colección, por ejemplo, `IRetribucionMadridService`, extrae de base de datos el resultado de la consulta construida con los parámetros indicados por `DialogFlow`. Busca en el fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto, y construye los dos ficheros, de texto y Excel, para que el usuario solicite su descarga si lo considera necesario.
- *SelectWithWhere()*: Al igual que el método descrito anteriormente, a través de la lógica de negocio de la interfaz de cada colección, por ejemplo, `IRetribucionMadridService`, extrae de base de datos el resultado de la consulta construida con los parámetros indicados por `DialogFlow`. Busca en el fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto, y construye los dos ficheros, de texto y Excel, para que el usuario solicite su descarga si lo considera necesario.
- *SatisfactoryQuery()*: A través de la lógica de negocio de la interfaz `ILogService`, extrae de base de datos el último Log introducido para el usuario, y se actualiza el campo `consulta_satisfactoria`. Busca en el fichero de texto `Message.properties` el texto correspondiente a este intent para completar la salida de texto.

### 4.2.3 Capa de acceso a datos

Es la capa donde se encuentra los modelos de cada tabla de la base de datos, así como la lógica para el acceso a su información a través de la API JPA <sup>26</sup> y el driver JDBC <sup>27</sup>.

Sus códigos fuentes están en los paquetes: *es.uam.eps.tfg.ChatbotOpenData.model* y *es.uam.eps.tfg.ChatbotOpenData.repository*.

Respecto al paquete de los modelos, se ha añadido un modelo por cada tabla de la base de datos (ver 3.2.3), es decir, hay ocho modelos diferentes en nuestra aplicación. Haciendo uso de las anotaciones disponibles para Spring Data JPA, se pueden definir algunas características de la entidad:

---

<sup>26</sup> Accessing Data with JPA, <https://spring.io/guides/gs/accessing-data-jpa/>

<sup>27</sup> PostgreSQL 7.4.30 Documentation, <https://www.postgresql.org/docs/7.4/jdbc-use.html>

- *Entidad*: Se indica con la anotación `@Entity` que este objeto es una entidad JPA.
- *Tabla*: Se indica con la anotación `@Table(name = "nombre_tabla")` la tabla a la que hace referencia este modelo.
- *Id*: Anotada con `@Id`, JPA reconoce el atributo como la clave primaria de la entidad.
- *Columna*: Se indica con la anotación `@Column(name = "nombre_columna")` la columna a la que hace referencia este atributo.
- *Clave autogenerada*: Se indica con la anotación `@GeneratedValue(generator = sequence)` que el atributo es un valor autogenerado de la secuencia indicada.

Se han desarrollado trece interfaces para el acceso a datos en total. Para todos los modelos, se ha desarrollado una interfaz que extiende la interfaz de Spring Data JPA, `JpaRepository`, y que permite realizar las consultas básicas a la tabla de cada modelo. Y para los modelos `Keyword`, `Calendario_Madrid`, `Decide_Madrid`, `Presupuesto_General_Madrid` y `Retribución_Madrid`, también se ha desarrollado una interfaz para las consultas SQL más complejas, utilizando la interfaz `Statement` <sup>25</sup> y el objeto `ResultSet` <sup>26</sup>:

- *ICalendarioMadridDAO.java*: Se encarga de albergar la lógica de acceso a datos más compleja de la tabla `Calendario_Madrid`. Se establece conexión, utilizando la interfaz `Statement`, se ejecuta la consulta SQL, y devuelve los resultados, siendo leídos a través del objeto `ResultSet`.
- *IRetribucionMadridJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla `Calendario_Madrid`. Extiende la interfaz `JpaRepository`.
- *ICollectionJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla `Colección`. Extiende la interfaz `JpaRepository`.
- *IDecideMadridDAO.java*: Se encarga de albergar la lógica de acceso a datos más compleja de la tabla `Decide_Madrid`. Se establece conexión, utilizando la interfaz `Statement`, se ejecuta la consulta SQL, y devuelve los resultados, siendo leídos a través del objeto `ResultSet`.
- *IRetribucionMadridJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla `Decide_Madrid`. Extiende la interfaz `JpaRepository`.
- *IKkeywordDAO.java*: Se encarga de albergar la lógica de acceso a datos más compleja de la tabla `Keyword`. Se establece conexión, utilizando la interfaz `Statement`, se ejecuta la consulta SQL, y devuelve los resultados, siendo leídos a través del objeto `ResultSet`.
- *IKkeywordJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla `Keyword`. Extiende la interfaz `JpaRepository`.
- *ILogJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla `Log`. Extiende la interfaz `JpaRepository`.
- *IPresupuestoGeneralMadridDAO.java*: Se encarga de albergar la lógica de acceso a datos más compleja de la tabla `Presupuesto_General_Madrid`. Se establece conexión, utilizando la interfaz `Statement`, se ejecuta la consulta SQL, y devuelve los resultados, siendo leídos a través del objeto `ResultSet`.
- *IPresupuestoGeneralMadridJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla `Presupuesto_General_Madrid`. Extiende la interfaz `JpaRepository`.
- *IRetribucionMadridDAO.java*: Se encarga de albergar la lógica de acceso a datos más compleja de la tabla `Retribución_Madrid`. Se establece conexión, utilizando la interfaz `Statement`, se ejecuta la consulta SQL, y devuelve los resultados, siendo leídos a través del objeto `ResultSet`.

- *IRetribucionMadridJPA.java*: Se encarga de albergar la lógica de acceso a datos CRUD de la tabla Retribución\_Madrid. Extiende la interfaz JpaRepository.
- *IUserJPA.java*: Se encarga de albergar toda la lógica de acceso a datos de la tabla Usuario. Extiende la interfaz JpaRepository.

Se ha añadido un último paquete, *es.uam.eps.tfg.ChatbotOpenData.util*, donde se han desarrollado todas las utilidades necesarias para la aplicación, incluida la conexión con la base de datos.

Los datos abiertos son comúnmente publicados como archivos con datos estructurados en formatos estandarizados como CSV, XML o RDF. En este caso, los ficheros con los que se han trabajado son CSV. Los métodos y funcionalidades para poder operar con ellos también se han incluido en este paquete.

## 5 Integración, pruebas y resultados

---

En este capítulo de la memoria se va a mostrar cómo se ha realizado la integración del chatbot en la plataforma de mensajería Telegram y del servidor web en DialogFlow (ver Anexo D), y la evaluación del chatbot mediante un estudio de usuarios, comparando el uso de este con el uso del buscador y página web del portal.

### 5.1 Experimento

De la bibliografía consultada, únicamente Neumaier [5] presenta una evaluación sobre un chatbot de datos abiertos gubernamentales. Se realiza un estudio a siete participantes teniendo que realizar una tarea de búsqueda y rellenar un cuestionario sobre la eficiencia del sistema y expectativas del usuario.

Para diseñar nuestro experimento, se consideran los trabajos previos de evaluación de chatbots en el gobierno electrónico [6] y los trabajos sobre metodologías y métricas de evaluación de encuestas para sistemas conversacionales [7, 8].

#### 5.1.1 Metodología de evaluación

El experimento realizado consta de un estudio de usuarios en un entorno controlado. Los participantes fueron supervisados y sus acciones monitorizadas y grabadas. El estudio ha sido realizado bajo unas condiciones similares en todos los casos.

En el estudio participan doce personas, de manera libre y tras rellenar un formulario de consentimiento donde se les informa sobre la finalidad del experimento y condiciones:

- *Sexo*: Ocho hombres y cuatro mujeres.
- *Rangos de edad*: De 18 a 24 años, cuatro; de 25 a 34 años, cinco; de 35 a 44 años, uno; y de 45 a 54 años, dos.
- *Nivel de estudios*: Uno en educación básica obligatoria (ESO), tres en formación profesional, tres graduados, cuatro licenciados y un doctorado. Los participantes con estudios superiores habían estudiado: uno Ciencias, dos Ciencias Sociales, uno Artes y Humanidades y cuatro Ingenierías.
- *Experiencia en motor de búsquedas (Google, Yahoo...)*: Todos lo usan con frecuencia.
- *Experiencia en chatbots*: Todos han usado chatbots o asistentes virtuales alguna vez, excepto un usuario, aunque tenía conocimiento de ellos.
- *Experiencia con datos abiertos*: Solo un usuario tenía experiencia.
- *Experiencia manejo hojas de cálculo*: Dos nivel bajo, ocho nivel medio y dos nivel alto.
- *Experiencia manejo SQL*: Cinco nivel nulo, cinco nivel bajo y dos nivel alto.

El objetivo es evaluar el chatbot en comparación con el método tradicional a través del portal (buscar la colección deseada, descargar el documento, abrir y procesar el archivo).

Por este motivo, los participantes se dividen en dos grupos: un grupo utiliza el portal y la hoja de cálculo Excel (ver Anexo C); y el otro grupo el chatbot desarrollado (ver Anexo B). El estudio se centra en las seis colecciones (ver 3.1) extraídas del portal Datos Abiertos del Ayuntamiento de Madrid.

Tras completar el formulario de consentimiento y el cuestionario inicial anónimo para recuperar los datos personales, arriba expuestos, los participantes reciben las explicaciones sobre los datos abiertos y las hojas de cálculo o el chatbot, según el grupo correspondiente. Se les plantean tres tareas en dificultad creciente:

- *Tarea 1:* Encontrar los días festivos en 2021. La colección es “Calendario laboral”, y contiene los días laborables y no laborables oficiales desde el 1 de enero de 2013 al 31 de diciembre del 2021. Esta tarea implica dos filtros en dos campos diferentes (año 2021 y festivo).
- *Tarea 2:* Encontrar el nombre y salario de los diez concejales con mayor retribución en 2020. La colección objetivo es “Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid, del año 2020”, y contiene el salario mensual del alcalde, concejales y altos cargos del ayuntamiento en el año 2020. Esta tarea implica un filtro (concejal) y una ordenación y limitación del resultado obtenido.
- *Tarea 3:* Encontrar el dinero total asignado en el presupuesto 2020 para actividades culturales. La colección objetivo es “Presupuesto General. Año 2020”, y contiene la información del presupuesto gastado en la ciudad en 2020. Esta tarea implica un filtro (actividades culturales) y una función de agregación tras realizar una agrupación.

Tras la realización de cada tarea, los participantes rellenan un cuestionario sobre la dificultad de la tarea y la utilidad de las herramientas utilizadas. Tras la realización de todas las tareas, se solicita a los participantes un último cuestionario sobre la satisfacción de las herramientas utilizadas (ver sección 5.1.2).

<b>Id</b>	<b>Valor servicio público</b>	<b>Detalle</b>
<b>I1</b>	Efectividad	Ofrece soluciones a tareas como las propuestas
<b>I2</b>	Eficiencia	Realiza rápidamente tareas como las propuestas
<b>I3</b>	Apertura	Revela de forma transparente su identidad y la razón de su funcionamiento
<b>I4</b>	Equidad	Realiza un proceso justo, sin favorecer ni discriminar datos
<b>I5</b>	Profesionalidad	Sigue una conducta competente, respetuosa y coherente
<b>I6</b>	Legitimidad	Sigue unos procedimientos aparentemente legales y razonables desde el punto de vista del manejo y presentación de los datos
<b>I7</b>	Responsabilidad	Ofrece explicación sobre sus resultados, limitaciones y fallos
<b>I8</b>	Confianza	Es confiable en cuanto al manejo de información se refiere
<b>I9</b>	Orientación al usuario	Permite al usuario decidir y controlar las acciones a realizar
<b>I10</b>	Aceptabilidad	Es beneficioso (útil) para tareas como las planteadas
<b>I11</b>	Compromiso del usuario	Aumenta la satisfacción y el interés por realizar tareas como las planteadas
<b>I12</b>	Privacidad	Preserva la privacidad del usuario
<b>I13</b>	Adaptabilidad	Se adapta al dispositivo empleado: PC, tablet, teléfono móvil
<b>I14</b>	Inteligencia colaborativa	Ayuda a complementar habilidades del usuario y servicios externos en tareas como las planteadas

**Tabla 5-1: Items del cuestionario dirigido a evaluar los valores de los servicios públicos [9]**



### 5.1.1 Métricas de evaluación

Se ha llevado el estudio con métricas objetivas y subjetivas.

Por un lado, se registran todas las acciones de los participantes en los sistemas, tales como número de interacciones, marcas de tiempo o el grado de finalización de la tarea; estos se almacenan, y se analizan las métricas de efectividad y eficiencia.

Por otro lado, tras la finalización de cada una de las tareas, rellena un formulario sobre el grado de dificultad percibido en las tareas y la complejidad de las operaciones del sistema.

Además de estas métricas, consideramos el marco teórico propuesto por Makasi [9] destinado a establecer las características que un chatbot de servicio público debería tener. En concreto, se implementa como un cuestionario final con catorce cuestiones destinadas a evaluar si se cumplieron los valores públicos propuestos. Como se puede ver en la tabla 5-1, las cuestiones I1 a I8 están orientadas al sistema; las cuestiones I9 a I12 al usuario; e I13 e I14 a aspectos contextuales.

## 5.2 Resultados

Se presentan y discuten los resultados del estudio distinguiendo entre los dos grupos (seis participantes utilizan el chatbot, y los otros seis el portal del ayuntamiento) anteriormente descritos, analizando las medidas objetivas de eficiencia y eficacia, y las medidas subjetivas de usabilidad y valores de los servicios públicos.

### 5.2.1 Eficacia y eficiencia

Para medir ambas medidas en los dos sistemas evaluados, se registran todas las acciones realizadas por los participantes de las tres tareas planteadas en dos procesos: encontrar la colección de interés y obtener la solución solicitada a las consultas planteadas.

Los datos registrados y calculados son:

- Cálculo de porcentaje de finalización de la tarea, independientemente de si la solución es correcta.
- Tiempo medio en minutos.
- Promedio de número de interacciones con el sistema para llegar a la solución.

La tabla 5-2 y 5-3 muestra los valores de estas métricas:

- *Eficacia*: Usando el portal y la hoja de cálculo, para el primer proceso, encontrar la colección de interés, un 5.6% no fue completado, siendo el resto completado con éxito. En el segundo proceso, obtener la solución a la consulta planteada, el 11.1% no fue completado, 22.2% completado de manera incorrecta y un 66.6% completado de manera correcta. Haciendo uso del chatbot, en ambos procesos, el 100% lo completa de manera correcta.
- *Eficiencia*: Haciendo uso del portal y la hoja de cálculo, se ha observado un aumento de tiempo de uso y número de interacciones respecto al chatbot. En el primer proceso, se invierte un tiempo medio de 1.5 minutos, 3.1 interacciones con el sistema y 1.3 colecciones visitadas usando el portal, frente a 0.5 minutos, 2.3 interacciones y 1 colección visitada usando el chatbot. En el segundo proceso, se invierte 5.8 minutos y 8.1 operaciones para completar la tarea usando la hoja de cálculo, frente a 3.3 minutos y 2.3 operaciones usando el chatbot. El tiempo total invertido en ambos procesos por el portal y la hoja de cálculo casi duplica al invertido en el chatbot, 7.3 minutos frente a 3.8 minutos.

<b>PORTAL</b>					
<b>Encontrar la colección de interés</b>					
<b>Tarea</b>	<b>% Completado correctamente</b>	<b>% Completado incorrectamente</b>	<b>Media tiempo</b>	<b>Media interacciones</b>	<b>Media colecciones</b>
<b>1</b>	100.0	0.0	1.5	3.7	1.0
<b>2</b>	100.0	0.0	1.3	2.2	1.5
<b>3</b>	83.3	0.0	1.7	3.5	1.5
<b>Media</b>	<b>94.4</b>	<b>0.0</b>	<b>1.5</b>	<b>3.1</b>	<b>1.3</b>
<b>Obtener la solución a la consulta</b>					
<b>Tarea</b>	<b>% Completado correctamente</b>	<b>% Completado incorrectamente</b>	<b>Media tiempo</b>	<b>Media operaciones</b>	<b>Media tiempo total</b>
<b>1</b>	83.3	16.7	3.3	2.2	4.8
<b>2</b>	83.3	16.7	5.9	5.5	7.2
<b>3</b>	33.3	33.3	8.2	16.8	9.9
<b>Media</b>	<b>66.7</b>	<b>22.2</b>	<b>5.8</b>	<b>8.1</b>	<b>7.3</b>

Tabla 5-2: Valores de eficacia y eficiencia para las tareas en el portal y hoja de cálculo

<b>CHATBOT</b>					
<b>Encontrar la colección de interés</b>					
<b>Tarea</b>	<b>% Completado correctamente</b>	<b>% Completado incorrectamente</b>	<b>Media tiempo</b>	<b>Media interacciones</b>	<b>Media colecciones</b>
<b>1</b>	100.0	0.0	0.5	2.8	1.0
<b>2</b>	100.0	0.0	0.5	2.2	1.0
<b>3</b>	100.0	0.0	0.5	1.8	1.0
<b>Media</b>	<b>100.0</b>	<b>0.0</b>	<b>0.5</b>	<b>2.3</b>	<b>1.0</b>
<b>Obtener la solución a la consulta</b>					
<b>Tarea</b>	<b>% Completado correctamente</b>	<b>% Completado incorrectamente</b>	<b>Media tiempo</b>	<b>Media operaciones</b>	<b>Media tiempo total</b>
<b>1</b>	100.0	0.0	2.9	2.3	3.4
<b>2</b>	100.0	0.0	2.8	2.2	3.3
<b>3</b>	100.0	0.0	4.1	2.5	4.6
<b>Media</b>	<b>100.0</b>	<b>0.0</b>	<b>3.3</b>	<b>2.3</b>	<b>3.8</b>

Tabla 5-3: Valores de eficacia y eficiencia para las tareas en el chatbot

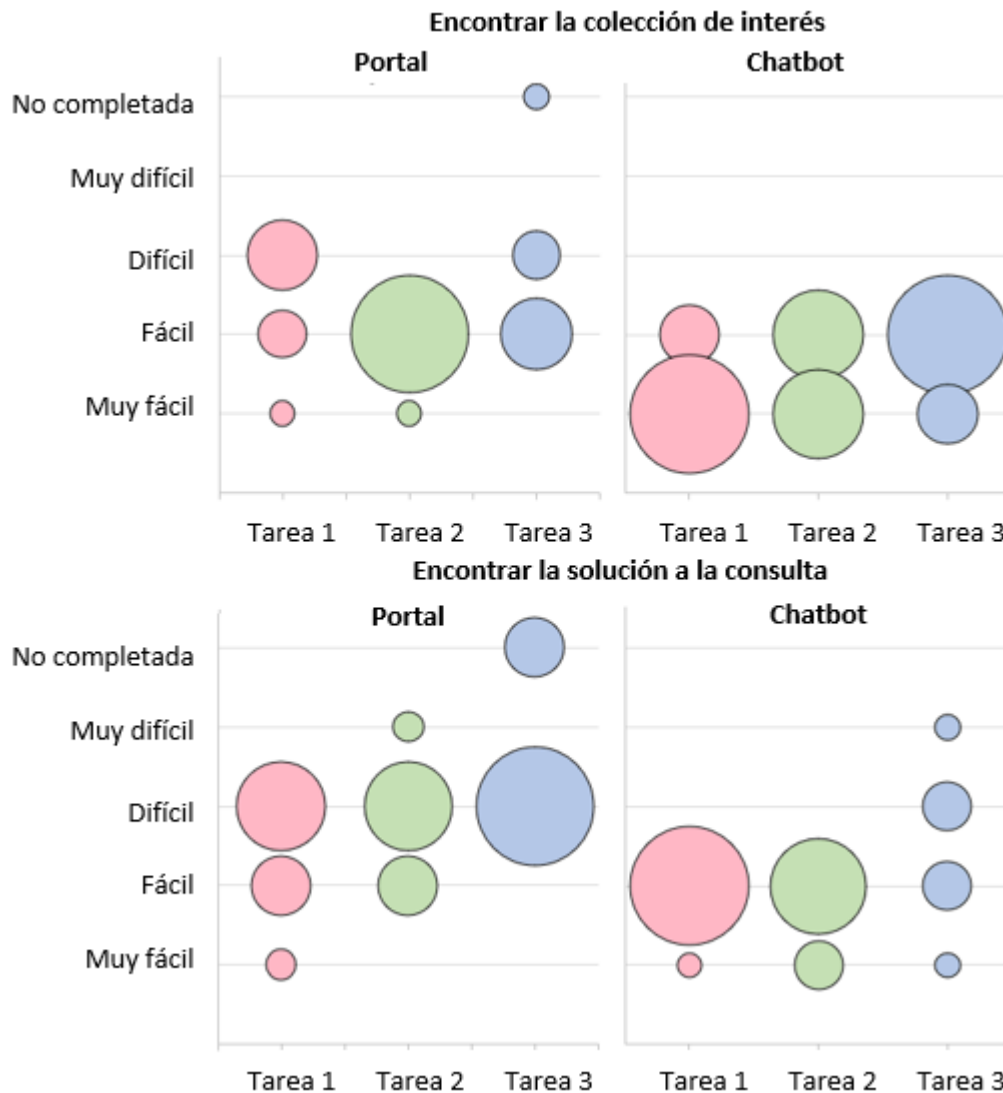
Las principales complicaciones descritas por los participantes en el portal fue la dificultad a la hora de buscar la colección deseada a través de palabras claves, sin aceptar sinónimos ni corregir errores ortográficos.

En el segundo proceso, los participantes con mayor conocimiento y experiencia en el uso de las hojas de cálculo mostraron mayor porcentaje de éxito en la realización de las tareas respecto a los que no. Esto no se observa en el chatbot, ya que el conocimiento en SQL no influencia en un uso más eficiente del chatbot. Esto evidencia la usabilidad del chatbot, independientemente de la experiencia con bases de datos y SQL.

### 5.2.2 Usabilidad

Tras la realización de cada tarea, los participantes rellenan un cuestionario donde califican la usabilidad de los sistemas utilizados. En este cuestionario, se pregunta también sobre lo intuitiva que es la interfaz de los sistemas.

En la figura 5-1 se muestra un gráfico de burbuja que representa el nivel de dificultad percibido (eje vertical) y las tareas (eje horizontal). Al igual que en la sección anterior, se ha dividido en dos procesos.



**Figura 5-1: Niveles de dificultad de las tareas percibidos por los participantes**

Se observa como los valores medidos y explicados en la sección anterior (ver 5.2.1) son apoyados por estos nuevos datos, donde los participantes que no usaron el chatbot, encontraron mayor dificultad que los que usaron el chatbot.

Todos los participantes están de acuerdo que las interfaces son fáciles de usar con los conocimientos adecuados y entrenamiento.

### 5.2.3 Valores de los servicios públicos

Estos valores, explicados en la sección 5.1.2, han sido recogidos a través de un cuestionario final a los participantes tras la realización de todas las tareas.

La figura 5-2 muestra un gráfico de barras que resume las respuestas recogidas. Se ha utilizado un código de colores para las diferentes respuestas: verde representa de acuerdo, amarillo indiferencia y rojo desacuerdo. Se analizan los valores para los dos sistemas:

- *Efectividad y eficiencia*: El chatbot tiene un acuerdo generalizado, sin embargo, el portal muestra menos de un 50% de apoyo. Ha sido analizado en la sección 5.2.1.
- *Apertura y responsabilidad*: El chatbot tiene opiniones más positivas que el portal en estos dos aspectos. Según los comentarios de los participantes, se debe a la intención de ayuda y explicaciones dadas por el chatbot en caso de error. También se remarca la incapacidad del motor de búsqueda del portal para recuperar las colecciones deseadas.
- *Confianza y legitimidad*: Ambos sistemas mostraron su conformidad con estos aspectos.
- *Adaptabilidad*: El 60% usuarios remarcaron que las hojas de cálculo en dispositivos más pequeños, como móviles, sería demasiado tedioso.

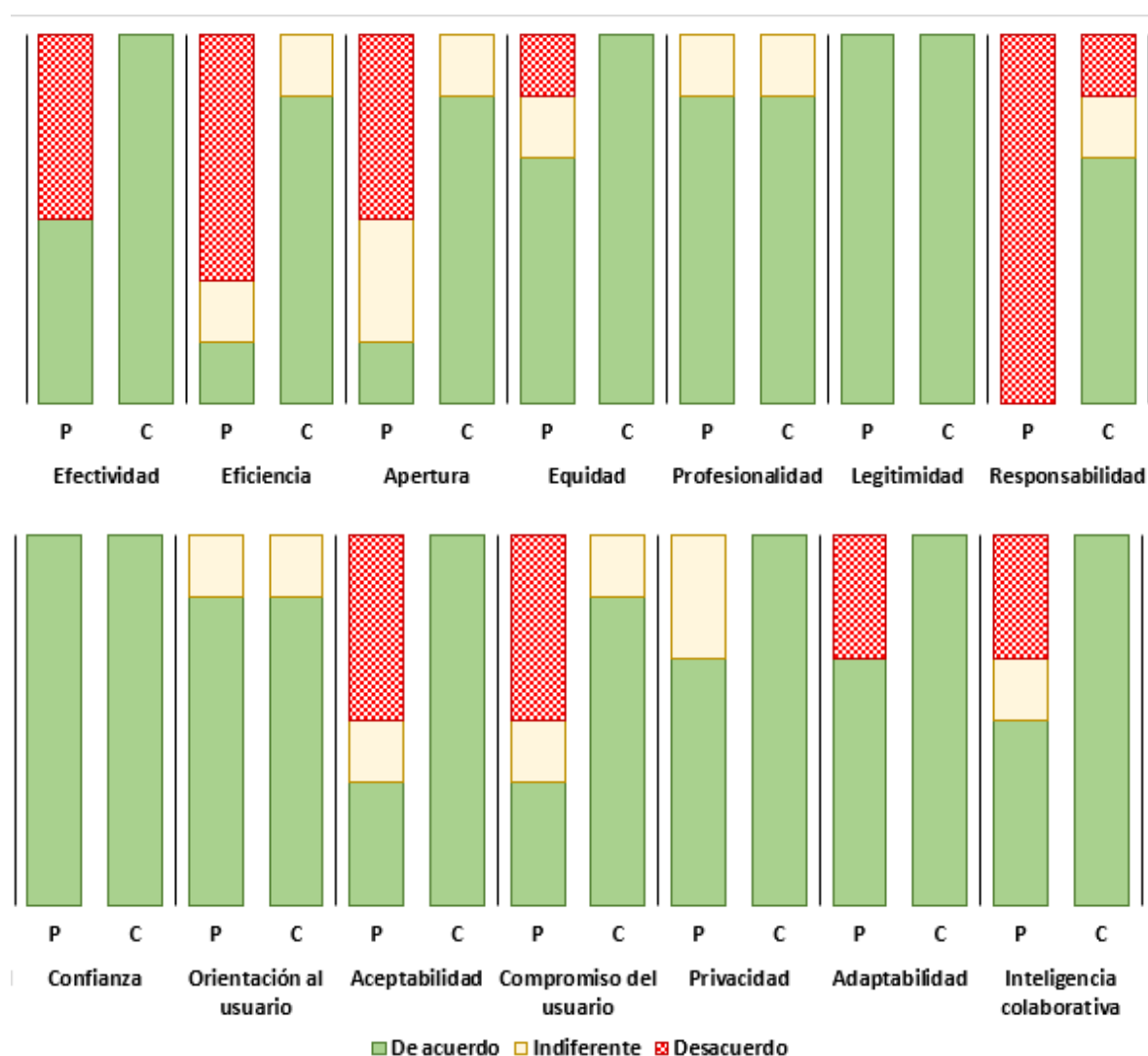


Figura 5-2: Opinión sobre los valores de los servicios públicos del portal (P) y el chatbot (C)

## 6 Conclusiones y trabajo futuro

---

### 6.1 Conclusiones

El trabajo desarrollado en esta memoria es un primer acercamiento al uso de chatbots para mejorar el acceso y uso de datos gubernamentales.

Se ha realizado un estudio en el ámbito actual del papel de los chatbots, de la cada vez más clara posición de los gobiernos en cumplir las políticas de Gobierno Abierto, y la relación entre estos dos puntos, y como esta permite a los ciudadanos hacer uso de los datos.

El estudio realizado ha encontrado evidencias sobre los beneficios del chatbot desarrollado respecto al portal y métodos de acceso a datos tradicionales. La principal limitación de la investigación se centra en la falta de diferentes contextos y en un reducido número de participantes y tareas. Una investigación futura debería extenderse a más participantes, tareas y escenarios no tan controlados como el llevado a cabo en este estudio. Además, todos los participantes deberían utilizar y probar tanto el chatbot como el portal para una posterior comparación entre ambos sistemas.

El sistema tiene como objetivo centrarse en no sólo ciudadanos no expertos, sino también a profesionales como gestores públicos y empresariales. Con este objetivo, se visualiza la necesidad de incorporar análisis y visualización previa de datos.

### 6.2 Trabajo futuro

Como se ha comentado en las conclusiones, se han observado varios puntos a continuar desarrollando.

En primer punto, la necesidad de seguir mejorando el buscador de colecciones y que permita la recuperación de información a través de la indexación de los términos, funcionalidad que utilizan los buscadores webs actuales; por ejemplo, la API Lucene.

Continuar ampliando las posibles consultas que el chatbot permite, añadiendo las cláusulas HAVING o JOIN.

Con el fin de seguir mejorando la interacción con el público, y con un estudio con más participantes, añadir más frases de entrenamientos para activar los diferentes intents, además de hacer el chatbot multidioma. Se ha observado que el Webhook Request (ver Anexo A) tiene entre la información de Telegram el idioma que el usuario tiene configurado en la aplicación. Tras analizar este valor, que el chatbot cambie el idioma de interacción con el usuario.

Por último, y con el fin de ser totalmente operativo en un entorno real, continuar añadiendo todas las colecciones que hay en el portal del Ayuntamiento de Madrid.



## Referencias

---

- [1] Daniel Lathrop, Laurel Ruma, “Open government: Collaboration, transparency, and participation in practice”, O’Reilly Media, Inc, 2010.
- [2] Iván Cantador, María E Cortés-Cediel, Miriam Fernández. “Exploiting Open Data to analyze discussion and controversy in online citizen participation”, 2020.
- [3] Thomas L. Floyd, Prentice Hall, “Fundamentos de Sistemas Digitales”, Prentice Hall 9a Ed, 2006.
- [4] Vito Walter Anelli, Tommaso Di Noia, Eugenio Di Sciascio, Azzurra Ragone, “Anna: A virtual assistant to interact with Puglia Digital Library (discussion paper).”, 2019.
- [5] Sebastian Neumaier, Vadim Savenkov, Svitlana Vakulenko, “Talking open data” In 14th European Semantic Web Conference. Springer, 2017, pp. 132–136.
- [6] Simone Porreca, Francesco Leotta, Massimo Mecella, Stavros Vassos, Tiziana Catarci, “Accessing government open data through chatbots.” In Proceedings of the 17th International Conference on Web Engineering. Springer, 2017, pp. 156–165.
- [7] Dijana Peras, “Chatbot evaluation metrics.”, In Proceedings of the 36th International Scientific Conference on Economic and Social Development, Varazdin Development and Entrepreneurship Agency, 2018, pp. 89–97.
- [8] Simone Porreca, Francesco Leotta, Massimo Mecella, Tiziana Catarci, “Chatbots as a novel access method for government open data.”, In Proceedings of the 25th Italian Symposium on Advanced Database Systems, CEUR Workshop Proceedings, 2017, pp. 122–129.
- [9] Wari Maroengsit, Thanarath Piyakulpinyo, Korawat Phonyiam, Suporn Pongnumkul, Pimwadee Chaovalit, Thanaruk Theeramunkong, “A survey on evaluation methods for chatbots.”, In Proceedings of the 2019 7th International Conference on Information and Education Technology, ACM, 2019, pp. 111–119.





# Glosario

---

API	Abreviatura de Application Programming Interface. Procesos, funciones y métodos que puede ser empleada por otro programa desde una capa de abstracción.
CRUD	Acrónimo de Create, Read, Update and Delete, que se usa para referirse a las funciones básicas de la persistencia de base de datos.
Framework	Conjunto estandarizado de criterios y conceptos para enfocar un problema particular como referencia, para futuros problemas de índole similar.
Gobierno Abierto	Doctrina política que tiene como objetivo que los ciudadanos colaboren en la creación y la mejora de los servicios públicos y en el robustecimiento de la transparencia y la rendición de cuentas.
Heroku	Plataforma que permite construir, entregar, supervisar y alojar aplicaciones en la nube.
Heroku Postgres	Servicio de base de datos SQL administrado por Heroku.
JSON	Formato de texto para el intercambio de datos.
Log	Registro de los acontecimientos que han afectado a un sistema informático de manera cronológica.
Maven	Herramienta de software para la gestión y construcción de proyectos Java, que utiliza un POM con las dependencias de los módulos y componentes externos, para describir el proyecto.
POST	Método HTTP utilizado para enviar los datos usando una URL de manera oculta para el usuario.
Spring Boot	Framework utilizado para desarrollar aplicaciones web en el lenguaje Java. Centra sus esfuerzos en el desarrollo debido a que el diseño abstrae la configuración y los detalles de bajo nivel.
Webhook Request	Solicitud POST HTTPS de DialogFlow al servicio web. El cuerpo de esta solicitud es un objeto JSON con información sobre el intent que coincide.
Webhook Response	Respuesta HTTPS enviada desde el servicio web a DialogFlow. El cuerpo de esta solicitud es un objeto JSON con la respuesta.



# Anexos

---

## A Detalle Webhook Request y Webhook Response

Para establecer la comunicación entre DialogFlow y el servidor web, se debe aplicar un servicio de Webhook, que acepte y envíe solicitudes JSON.

Este servicio Webhook debe cumplir con una serie de requisitos:

- Debe administrar solicitudes HTTPS.
- La URL para las solicitudes debe ser de carácter público.
- Debe administrar las solicitudes POST con un cuerpo JSON Webhook Request.
- Debe responder a estas solicitudes con un cuerpo JSON Webhook Response.

Las siguientes figuras muestran ambas solicitudes, correspondientes al intent Filter Keywords Collections.

### Webhook Request

```
{
  "originalDetectIntentRequest": {
    "payload": {
      "data": {
        "text": "2020",
        "date": 1610984367,
        "message_id": 4866,
        "chat": {
          "type": "private",
          "id": "911558041"
        },
        "from": {
          "language_code": "es",
          "id": 911558041,
          "first_name": "Jesus"
        }
      },
      "source": "telegram"
    },
    "queryResult": {
      "allRequiredParamsPresent": true,
      "fulfillmentMessages": [
        {
          "text": {
            "text": [
              "Error al conectar con el servidor."
            ]
          }
        }
      ],
      "fulfillmentText": "Error al conectar con el servidor.",
      "intent": {
        "displayName": "Filter Keywords Collections",
        "name": "projects/cursor1-tiphgs/agent/intents/ac9a9f09-d852-4ea0-8c33-ee38863998dd"
      },
      "intentDetectionConfidence": 0.3,
      "languageCode": "es",
      "outputContexts": [
        {
          "name": "projects/cursor1-tiphgs/agent/sessions/05fa7d22-ccf3-3979-892d-9141674b683e/contexts/_system_counters_",
          "parameters": {
            "no-input": 0,
            "no-match": 0,
            "keywords": "2020",
            "keywords.original": "2020"
          }
        }
      ],
      "parameters": {
        "keywords": "2020"
      },
      "queryText": "2020"
    },
    "responseId": "c4d5a3c5-525e-4bbf-ac27-86158a78212c-842d41f9",
    "session": "projects/cursor1-tiphgs/agent/sessions/05fa7d22-ccf3-3979-892d-9141674b683e"
  }
}
```

Cuando en DialogFlow haya una coincidencia con un intent, se envía una solicitud POST HTTPS al servicio web. Este cuerpo contiene la información del intent:

- *OriginalDetectIntentRequest*: Representa el contenido de la fuente de la solicitud original.
  - *Payload*: Representa la información original sin tratamiento de DialogFlow. Se puede observar el texto introducido “2020”, la fecha del mensaje “1610984367”, el identificador del mensaje, y la información del usuario que ha escrito el mensaje, en este caso “Jesus”, con el idioma “es” y el identificador del usuario “911558041”.
  - *Source*: Representa el servicio que utilizó el usuario para mantener la conversación “telegram”.
- *QueryResult*: Representa el resultado de la consulta conversacional una vez ya ha sido tratada por DialogFlow.
  - *Fulfillment\_messages* y *fulfillment\_text*: Representa el mensaje o los mensajes por defecto que DialogFlow mostrará al usuario en caso de que no se establezca conexión con el servidor web.
  - *Intent*: Representa el intent que coincide con la consulta conversacional “Filter Keywords Collections”.
  - *Parameters*: Representa la colección de parámetros extraídos. El parámetro es “keywords” con un valor de “2020”.
- *ResponseId*: Representa el identificador de la respuesta.
- *Session*: Representa el identificador de la sesión de la solicitud.

## Webhook Response

```
{
  "fulfillmentMessages": [
    {
      "payload": {
        "telegram": {
          "parse_mode": "html",
          "text": "<b>Ha filtrado por las palabras claves:
                </b> \"2020\"\\n\\nEstas son las colecciones que cumplen su filtro:\\n
                <pre><b>3</b></pre> <i>Retribuciones de concejales, directivos y eventuales
                del Ayuntamiento de Madrid. Año 2020</i>\\n\\n<b>6</b></pre> <i>Presupuesto General.
                Año 2020</i>\\n\\n</pre>\\n\\n"
        }
      }
    },
    {}
  ]
}
```

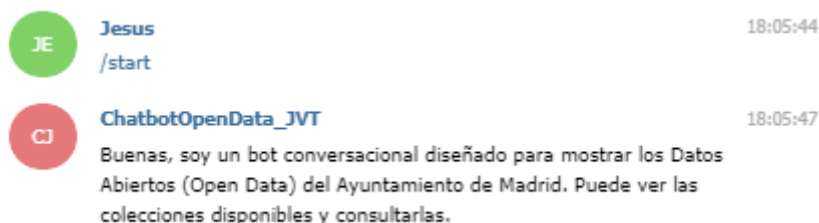
Tras recibir la solicitud, se debe enviar una respuesta de webhook. El cuerpo de esta respuesta tiene diferentes campos dependiendo del tipo de respuesta que se vaya a enviar; en este caso una respuesta de texto. Este cuerpo contiene la información del intent:

- *Fulfillment\_messages*: Representa el mensaje de respuesta enriquecida. Se puede observar que es destinado a “telegram”, utiliza etiquetas “html” y contiene un texto.

## B Ejemplo de conversación con el bot

Esta sección incluye una conversación entre un usuario y el bot, en el que el usuario va a realizar las interacciones para activar todos los intents (ver 4.1). En la sección 3.2.1 se puede observar una descripción de cada intent (ver tabla 3-1) y sus frases de entrenamiento (ver tabla 3-2):

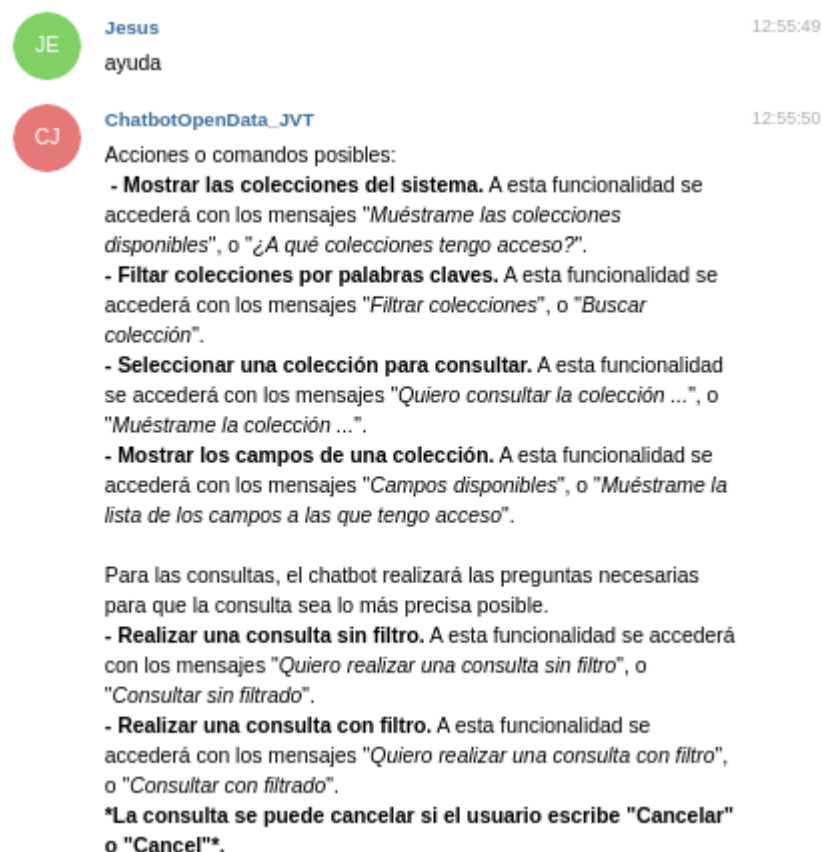
1. El usuario inicia la conversación con el bot (ver 4.1.1).



**Jesus** 18:05:44  
/start

**ChatbotOpenData\_JVT** 18:05:47  
Buenas, soy un bot conversacional diseñado para mostrar los Datos Abiertos (Open Data) del Ayuntamiento de Madrid. Puede ver las colecciones disponibles y consultarlas.

2. El usuario solicita ayuda al bot, a través del mensaje “Necesito ayuda”. El bot responde con las diferentes acciones que puede reconocer (ver 4.1.2).



**Jesus** 12:55:49  
ayuda

**ChatbotOpenData\_JVT** 12:55:50  
Acciones o comandos posibles:  
- **Mostrar las colecciones del sistema.** A esta funcionalidad se accederá con los mensajes "Muéstrame las colecciones disponibles", o "¿A qué colecciones tengo acceso?".  
- **Filtrar colecciones por palabras claves.** A esta funcionalidad se accederá con los mensajes "Filtrar colecciones", o "Buscar colección".  
- **Seleccionar una colección para consultar.** A esta funcionalidad se accederá con los mensajes "Quiero consultar la colección ...", o "Muéstrame la colección ...".  
- **Mostrar los campos de una colección.** A esta funcionalidad se accederá con los mensajes "Campos disponibles", o "Muéstrame la lista de los campos a las que tengo acceso".  
  
Para las consultas, el chatbot realizará las preguntas necesarias para que la consulta sea lo más precisa posible.  
- **Realizar una consulta sin filtro.** A esta funcionalidad se accederá con los mensajes "Quiero realizar una consulta sin filtro", o "Consultar sin filtrado".  
- **Realizar una consulta con filtro.** A esta funcionalidad se accederá con los mensajes "Quiero realizar una consulta con filtro", o "Consultar con filtrado".  
\*La consulta se puede cancelar si el usuario escribe "Cancelar" o "Cancel"\*.

3. El usuario solicita las colecciones que puede consultar a través del mensaje “mostrar colecciones” (ver 4.1.3).

**JE** **Jesus** 12:53:25  
mostrar colecciones

**CJ** **ChatbotOpenData\_JVT** 12:53:27  
Estas son las colecciones que gestiona el ChatBot. Por favor, indique el número para acceder a la información:

- 1) *Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2018*
- 2) *Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2019*
- 3) *Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2020*
- 4) *Calendario laboral*
- 5) *Decide Madrid*
- 6) *Presupuesto General. Año 2020*

4. El usuario realiza una búsqueda para filtrar las colecciones según la palabra clave “retribucion”. El usuario observa que hay tres colecciones que satisfacen este filtro (ver 4.1.4).

**JE** **Jesus** 12:53:43  
filtrar colecciones

**CJ** **ChatbotOpenData\_JVT** 12:53:44  
Enumere, separando por comas, las keywords de la colección que quiera buscar.

**JE** **Jesus** 12:53:54  
retribucion

**CJ** **ChatbotOpenData\_JVT** 12:53:55  
**Ha filtrado por las palabras claves: "RETRIBUCION"**

Estas son las colecciones que cumplen su filtro:

- 1) *Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2018*
- 2) *Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2019*
- 3) *Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2020*

5. El usuario selecciona la colección con identificador 3 a través del texto “Quiero ver la coleccion 3”. El usuario ya puede realizar consultas (ver 4.1.5).

**JE** **Jesus** 12:57:02  
quiero ver la coleccion 3

**CJ** **ChatbotOpenData\_JVT** 12:57:03  
Ha seleccionado la colección "**Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2020**".  
¿Qué acción quiere realizar?

6. El usuario consulta los campos que forma la colección (ver 4.1.6).

JE

**Jesus**

mostrar los campos que puedo consultar

13:00:08

CJ

**ChatbotOpenData\_JVT**

Estas son los campos que tiene la coleccion "**Retribuciones de concejales, directivos y eventuales del Ayuntamiento de Madrid. Año 2020**".

13:00:09

- 1) Denominacion colectiva del puesto
- 2) Unidad de adscripcion
- 3) Numero de puesto
- 4) Denominacion individual del puesto
- 5) Nombre y apellidos
- 6) Fecha de acuerdo
- 7) Numero de boam
- 8) Fecha de boam
- 9)Codigo de cargo
- 10) Regimen dedicacion en enero
- 11) Retribucion bruta en enero
- 12) Regimen dedicacion en febrero
- 13) Retribucion bruta en febrero
- 14) Regimen dedicacion en marzo
- 15) Retribucion bruta en marzo
- 16) Regimen dedicacion en abril
- 17) Retribucion bruta en abril
- 18) Regimen dedicacion en mayo
- 19) Retribucion bruta en mayo
- 20) Regimen dedicacion en junio
- 21) Retribucion bruta en junio
- 22) Regimen dedicacion en julio
- 23) Retribucion bruta en julio
- 24) Regimen dedicacion en agosto
- 25) Retribucion bruta en agosto
- 26) Regimen dedicacion en septiembre
- 27) Retribucion bruta en septiembre
- 28) Regimen dedicacion en octubre
- 29) Retribucion bruta en octubre
- 30) Regimen dedicacion en noviembre
- 31) Retribucion bruta en noviembre
- 32) Regimen dedicacion en diciembre
- 33) Retribucion bruta en diciembre
- 34) Retribucion media

7. El usuario realiza la consulta con filtro “Mostrar los 5 nombres de los concejales con mayores sueldos”. Tras ver el resultado de la consulta, el usuario introduce su conformidad con el resultado de la consulta (ver 4.1.8 y 4.1.9).

JE	<b>Jesus</b> quiero consultar filtrando	13:00:38
CJ	<b>ChatbotOpenData_JVT</b> ¿Qué campos quiere consultar? Enumera los identificadores de los campos. Introduzca "Todo" si quiere consultar todos los campos.	13:00:38
JE	<b>Jesus</b> 5, 34	13:00:44
CJ	<b>ChatbotOpenData_JVT</b> ¿Qué filtros quiere aplicar? Enumera los identificadores de los campos, añadiendo el tipo de filtro ("<" menor / "<=" menor o igual / ">" mayor / ">=" mayor o igual / "=" igual / "!=" distinto) y su valor. Por ejemplo: "5 = Carmena, 33 > 5000".	13:00:44
JE	<b>Jesus</b> 4 = concejal	13:01:09
CJ	<b>ChatbotOpenData_JVT</b> ¿Qué funciones de agrupación quiere aplicar? Introduzca la función (SUM, MAX, MIN, AVG, COUNT) y el identificador del campo. Por ejemplo: "sum 5". Introduzca "No" si no quiere agrupar la consulta.	13:01:09
JE	<b>Jesus</b> no	13:01:13
CJ	<b>ChatbotOpenData_JVT</b> ¿Por qué campo quiere ordenarla? Introduzca el identificador y el orden (ASC, DESC). Por ejemplo: "4 desc". Introduzca "No" si no quiere ordenar la consulta.	13:01:13
JE	<b>Jesus</b> 34 desc	13:01:21
CJ	<b>ChatbotOpenData_JVT</b> ¿Cuántos registros quiere que se muestren? Introduzca el número de registros a devolver. Introduzca "No" si quiere que se devuelvan todos los resultados.	13:01:21
JE	<b>Jesus</b> 5	13:01:27



CJ	ChatbotOpenData_JVT	13:01:28
	<p><b>Ha consultado los campos:</b> nombre y apellidos, retribucion media; <b>de la colección 3; filtrando por:</b> denominacion individual del puesto = "CONCEJAL"; <b>sin función agrupación; ordenando los resultados por</b> retribucion media de manera DESC y <b>limitando el</b> número de resultados a 5.</p> <p>Estos son los campos que ha solicitado:</p> <p>C.P: Denominacion colectiva del puesto  U.O: Unidad de adscripcion  N.P: Numero de puesto  D.P: Denominacion individual del puesto  N.A: Nombre y apellidos  F.A: Fecha de acuerdo  N.B: Numero de boam  F.B: Fecha de boam  C.C:Codigo de cargo  R.D: Regimen dedicación mensual  R.B: Retribución bruta mensual  R.M: Retribución media</p> <p>N.A: MARTIN CASARIEGO CORDOBA  R.M: 8299.8</p> <p>N.A: LORETO SORDO RUIZ  R.M: 8299.8</p> <p>N.A: MARIA CAYETANA HERNANDEZ DE LA RIVA  R.M: 8299.8</p> <p>N.A: FRANCISCO JAVIER RAMIREZ CARO  R.M: 8299.8</p> <p>N.A: ALBERTO SERRANO PATINO  R.M: 6771.48</p> <p>Por favor, pulse para descargar el fichero de texto</p> <p>Por favor, pulse para descargar el fichero Excel</p>	
	¿Ha sido satisfactorio el resultado enviado?	13:01:28
JE	Jesus	13:02:21
	si	
CJ	ChatbotOpenData_JVT	13:02:22
	Muchas gracias por su respuesta.	

## C Ejemplo interacción con el portal

Esta sección incluye una interacción entre un usuario y el portal de datos abiertos del Ayuntamiento de Madrid, en el que el usuario va a realizar los pasos para consultar la colección “Calendario Laboral”. Se muestra también un ejemplo de datos abiertos de la colección elegida:

1. Se accede al portal a través de la URL indicada en la nota 23 (ver 3.1).
2. En el menú superior, accedemos a “Catálogo de datos”, tercer botón.
3. El portal tiene actualmente 476 colecciones, por lo que se usa el buscador para poder filtrar la colección elegida. Hay cuatro campos para filtrar, como se puede ver en la siguiente imagen.

En portada | Acerca de Datos Abiertos | Catálogo de datos | Colabora

Lo más visto ⓘ Tráfico. Cámaras / Tráfico Calle 30. Cámaras / Zonas del Servicio de Estacionami...

🏠 Catálogo de datos > Conjuntos de datos

🔍 🗨️ 📄 a+ a- 🖨️

### Conjuntos de datos

Conjuntos de datos

API

Listado con el catálogo completo de conjuntos de datos que el Ayuntamiento de Madrid pone a disposición de la ciudadanía. Este listado puede descargarlo en formato [RDF](#), ([DCAT](#)) o [CSV](#). De cada dataset se presenta su sector, la fecha en la que se incorporó al catálogo, su periodicidad, el número de descargas totales y los formatos en los que está disponible.

Se pueden realizar filtros utilizando los elementos que se despliegan en "Filtrar por..." para limitar los resultados a ciertos **sectores**, **formatos** y **periodicidades**, además de ordenar por nombre y fecha de incorporación.

#### Filtrar por...

Nombre o descripción

calendario laboral

Sector

Selecione...

Formato

Selecione...

Frecuencia de actualización

Selecione...

Ordenar por

Seleccionar

Nombre

Filtrar 🔍

4. Una vez filtrada, seleccionamos la colección.
5. Cada colección del portal posee una descripción sobre los datos que contiene, así como las etiquetas a las que pertenece esta colección, como el sector, fecha de incorporación y qué datos posee. Además, incluye una serie de enlaces con información relacionada, páginas de ayuda y una sección con comentarios de otros usuarios.

# Calendario laboral

[← Volver](#)

## Calendario laboral en el ámbito de la ciudad de Madrid

Recoge los días determinados en el Decreto de Consejo de Gobierno de la Comunidad de Madrid, más las dos fiestas locales de la ciudad de Madrid, 15 de mayo (San Isidro Labrador) y 9 de noviembre (Nuestra Señora de la Almudena).

Para acceder al Calendario oficial laboral de la Comunidad de Madrid, consulte la información disponible en el apartado Información relacionada.

### Definición del conjunto de datos

#### Sector

Sector público

#### Fecha de incorporación al catálogo:

31/01/2017

#### Datos obtenidos:

desde 01/01/2013 hasta 31/12/2021

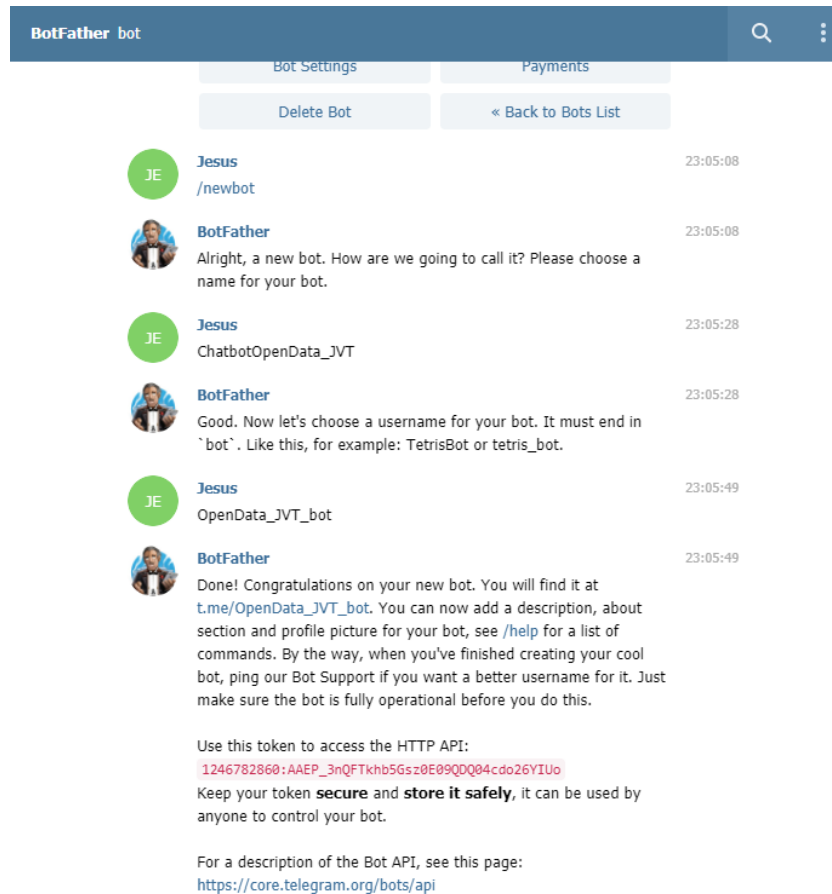
- Se procede a descargar los datos. En esta colección es posible hacerlo en los formatos XLS, TXT CSV y RDF. En este caso se descarga en formato XLS.
- Una vez descargada la colección en el formato elegido, el usuario procede a realizar la consulta deseada. En la siguiente imagen se adjuntan los primeros 17 de los 3288 registros totales de la colección.

	A	B	C	D	E
1	Día	Día_semana	laborable / festivo / domingo festivo	Tipo de Festivo	Festividad
2	01/01/2013	martes	festivo	Festivo nacional	Año Nuevo
3	02/01/2013	miércoles	laborable		
4	03/01/2013	jueves	laborable		
5	04/01/2013	viernes	laborable		
6	05/01/2013	sábado	sábado		
7	06/01/2013	domingo	domingo		
8	07/01/2013	lunes	festivo	Festivo nacional	Traslado Epifanía del Señor
9	08/01/2013	martes	laborable		
10	09/01/2013	miércoles	laborable		
11	10/01/2013	jueves	laborable		
12	11/01/2013	viernes	laborable		
13	12/01/2013	sábado	sábado		
14	13/01/2013	domingo	domingo		
15	14/01/2013	lunes	laborable		
16	15/01/2013	martes	laborable		
17	16/01/2013	miércoles	laborable		
18	17/01/2013	jueves	laborable		

## D Integración en Telegram

Tras desarrollar la funcionalidad de DialogFlow, declarar el entitie, los nueve intents, indicar sus contextos, frases de entrenamiento, respuestas y parámetros, el agente debe integrar la plataforma de mensajería Telegram.

Para ello, se debe utilizar BotFather, un bot propio de Telegram que permite crear y administrar los bots ya existentes del usuario. Tras introducir los campos que indica, obtenemos el enlace para acceder a su conversación, y el token que identifica a este.



Una vez obtenido el token, se accede a la consola de DialogFlow, a la sección Integrations, y buscamos y activamos la opción Telegram. Tras introducir el token, Telegram y el agente de DialogFlow quedan integrados.

